Chapter 1

Image Reader

Functions

`load_data(data_dir)`

Loads the CIFAR-10 dataset.

Arguments

`data_dir` **(str)** : Directory of CIFAR-10 data batches.

Returns

- `x_train` (numpy array): Training data, shape [50000, 3072], dtype=np.float32.
- `y_train` (numpy array): Training labels, shape [50000,], dtype=np.int32.
- `x_test` (numpy array): Test data, shape [10000, 3072], dtype=np.float32.
- `y_test` (numpy array): Test labels, shape [10000,], dtype=np.int32.

`train_vaild_split(x_train, y_train, split_index=45000)`

Splits training data into new training and validation datasets.

Arguments

`x_train` **(array)** : Original training data, shape [50000, 3072].

`y_train` **(array)** : Original training labels, shape [50000,].

`split_index` **(int)** : Index for splitting (default 45000).

Returns

- `x_train_new` (array): New training data, shape [split_index, 3072].
- `y_train_new` (array): New training labels, shape [split_index,].
- `x_valid` (array): Validation data, shape [50000-split_index, 3072].
- `y_valid` (array): Validation labels, shape [50000-split_index,].

Usage Example

```
x_train, y_train, x_test, y_test = load_data("/path/to/cifar-10-batches-py/")
x_train_new, y_train_new, x_valid, y_valid = train_vaild_split(x_train, y_train)
```

Chapter 2

Image Augmentation

Functions

`parse_record(record, training)`

Parses a record into an image and applies data preprocessing.

Arguments

`record` **(array)** : An array of shape [3072,]. Represents one row of the x_* matrix.

`training` **(boolean)** : Indicates training mode.

Returns

- `image` (array): An array of shape [3, 32, 32].

`preprocess_image(image, training)`

Preprocesses a single image.

Arguments

`image` **(array)** : An array of shape [32, 32, 3].

`training` **(boolean)** : Indicates training mode.

Returns

- `image` (array): An array of shape [32, 32, 3], after preprocessing.

Data Preprocessing Steps

1. **Reshape and Transpose**: The input record is reshaped and transposed to form an image.
2. **Training Mode Preprocessing**:
    - Padding: Adds four extra pixels on each side.
    - Random Cropping: Randomly crops a [32, 32] section.
    - Random Horizontal Flip: Flips the image horizontally with 50% probability.
3. **Normalization**: Subtracts the mean and divides by the standard deviation of the pixels, channel-wise.

Chapter 3

Block Architecture

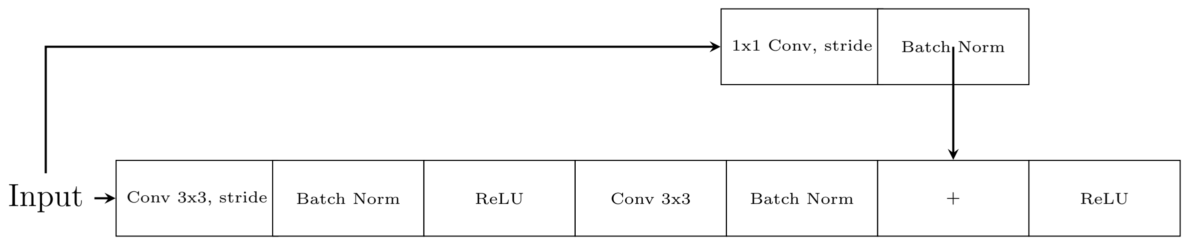
3.1 Standard Residual Block

The standard residual block (`standard_block`) in ResNet can include a projection shortcut when the input and output dimensions do not match. This projection shortcut is typically a 1x1 convolution applied to the input to match the dimensions of the output.

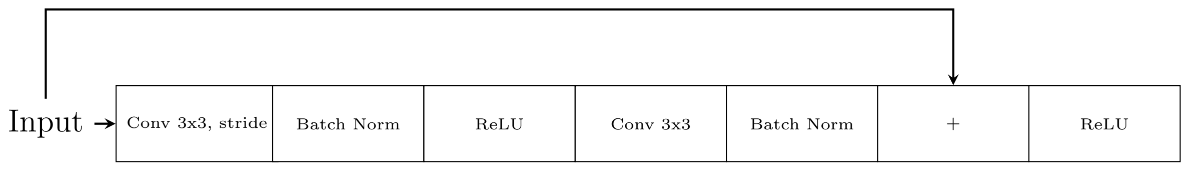
3.1.1 Structure with Projection Shortcut

- Convolutional Layer: Applies a 3x3 convolution.
- Batch Normalization and ReLU: Normalizes and applies the ReLU activation function.
- Convolutional Layer: Another 3x3 convolution.
- Batch Normalization: Final normalization step.
- Projection Shortcut (Optional): A 1x1 convolution applied to the input if dimension matching is required.
- Shortcut Connection: Adds the input (or projected input) to the block's output.

Standard Residual Block with Projection Shortcut



Standard Residual Block without Projection Shortcut



3.2 Bottleneck Block

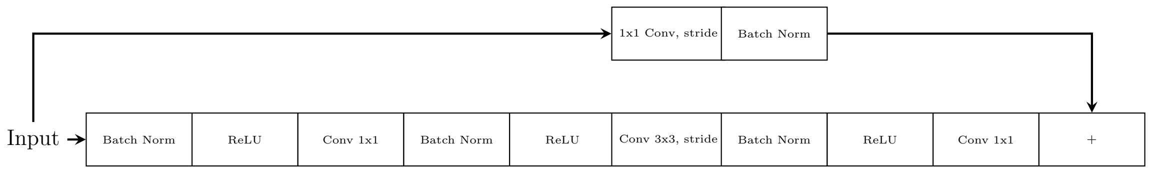
The bottleneck block (`bottleneck_block`) also includes a projection shortcut under the same conditions as the standard block.

3.2.1 Structure with Projection Shortcut

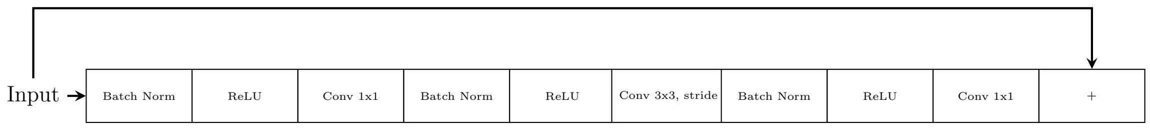
- Initial Batch Normalization and ReLU: Prepares the input.
- Convolutional Layer: Reduces dimensionality (1x1 convolution).
- Batch Normalization and ReLU: Normalization and activation.
- Convolutional Layer: Applies a 3x3 convolution.
- Batch Normalization and ReLU: Another round of normalization and activation.
- Convolutional Layer: Restores dimensionality (1x1 convolution).
- Projection Shortcut (Optional): A 1x1 convolution applied to the input if dimension matching is required.

- Shortcut Connection: Adds the input (or projected input) to the block's output.

Bottleneck Residual Block with Projection Shortcut



Bottleneck Residual Block without Projection Shortcut



3.3 stack_layer Class

The `stack_layer` class in ResNet architecture is responsible for creating a stack of residual blocks. These blocks can be either standard residual blocks or bottleneck blocks, forming the core building blocks of the ResNet model.

Initialization Parameters

- **filters**: The number of filters for the first convolution in each block.
- **block_fn**: A function that defines the type of block to be used, either `standard_block` or `bottleneck_block`.
- **strides**: The stride for the first block in the stack. If greater than 1, this layer will down-sample the input.
- **resnet_size**: The number of residual blocks in the stack layer.
- **first_num_filters**: The number of filters to use for the first block layer of the model.

Stack Construction

- The first block in the stack uses a projection shortcut and potentially a stride for down-sampling.
- Subsequent blocks in the stack do not use projection shortcuts and have a stride of 1, maintaining the dimensionality set by the first block.

Forward Pass

- The input tensor is passed sequentially through all the blocks in the stack.
- The output of each block is the input to the next block, culminating in the final output of the stack layer.

3.4 ResNet Output Layer

output_layer Class

The `output_layer` class in the ResNet architecture is designed to implement the final output layer of the network. This layer is responsible for producing the final classification results.

Initialization Parameters

- **filters**: The number of filters in the preceding layer.
- **resnet_version**: Specifies the version of ResNet (1 or 2). If set to 2, it indicates the use of bottleneck blocks.
- **num_classes**: The number of classes for classification.

Layer Components

- **Batch Normalization and ReLU (Optional)**: Applied only if `resnet_version` is 2, indicating the use of pre-activation in bottleneck blocks.

- **Global Average Pooling**: Reduces each feature map to a single value, effectively summarizing the spatial information.
- **Fully Connected Layer**: Transforms the pooled features into the final output for classification.

Forward Pass

- If `resnet_version` is 2, the input is first passed through batch normalization and ReLU.
- The input is then subjected to global average pooling.
- The pooled output is flattened and passed through a fully connected layer to produce the final classification result.

Chapter 4

ResNet Architecture

4.1 ResNet Class

The `ResNet` class is a PyTorch module that implements the ResNet architecture for image classification. It allows for the creation of different versions and sizes of ResNet models.

Initialization Parameters

- **resnet_version**: Specifies the version of ResNet (1 or 2). Version 2 uses bottleneck blocks.
- **resnet_size**: A positive integer representing the number of residual blocks in each stack layer.
- **num_classes**: The number of classes for classification.
- **first_num_filters**: The number of filters to use for the first block layer of the model. This number is doubled for each subsequent subsample block layer.

Architecture Components

- **Initial Convolution Layer**: Maps the 3 RGB channels to a specified number of channels (first_num_filters).

- **Batch Normalization and ReLU (Version 1)**: Applied after the initial convolution layer in ResNet version 1.
- **Stack Layers**: A series of stack layers, each containing multiple residual blocks (standard or bottleneck blocks).
- **Output Layer**: Processes the final output from the stack layers and produces the classification result.

Forward Pass

1. The input is first passed through the initial convolution layer.
2. If ResNet version 1 is used, the output of the initial layer is then passed through batch normalization and ReLU.
3. The output is sequentially passed through each stack layer.
4. Finally, the output from the stack layers is processed by the output layer to produce the classification logits.

Stack Layers

Each stack layer in the ResNet architecture consists of multiple residual blocks. The first block in each stack layer uses a projection shortcut and potentially a stride of 2 for downsampling. Subsequent blocks in the same stack layer do not use projection shortcuts and have a stride of 1.

Output Layer

The output layer includes global average pooling followed by a fully connected layer. It transforms the final feature maps into logits for classification.

Explanation of Image Size Reduction and Channel Increase in ResNet

In the ResNet class, the image size reduction and increase in the number of output channels are primarily managed within the loop that creates the stack layers. This process is crucial for the ResNet architecture as it allows the network to learn increasingly abstract features at different scales.

Loop for Creating Stack Layers

- The loop iterates three times, creating three stack layers. Each stack layer consists of multiple residual blocks (either standard or bottleneck blocks).
- **Input and Output Channels**:
  - In the first iteration ($i = 0$), the number of input channels for the first stack layer is set to `self.first_num_filters`.
  - For subsequent iterations ($i > 0$), the number of input channels is equal to the number of output channels from the previous stack layer.
  - The number of output channels for each stack layer depends on the ResNet version:
    - **Version 1**: The output channels are doubled with each stack layer. Specifically, `out_channel = self.first_num_filters * (2**i)`.
    - **Version 2 (Bottleneck)**: The output channels are quadrupled from the input channels of the same stack layer, i.e., `out_channel = in_channel * 4`.
- **Strides for Downsampling**:
  - For the first stack layer ($i = 0$), a stride of 1 is used, meaning no downsampling occurs in this layer.
  - For the subsequent stack layers ($i > 0$), a stride of 2 is used in the first block of each stack. This stride reduces the spatial dimensions (height

and width) of the feature maps by half, effectively downsampling the input.

- **Effect on Image Size and Channels**:
  - The combination of increasing the number of output channels and reducing the spatial dimensions allows the network to learn more complex features while reducing the computational load.
  - In bottleneck blocks (ResNet version 2), the increase in channels is more significant, allowing the network to learn even more complex features at each layer.

Chapter 5

Cifar Training, Validation, and Testing Class

The `Cifar` class is designed for training, validation, and testing of a ResNet model on the CIFAR dataset. It encapsulates the network initialization, training process, validation/testing, and checkpoint management.

5.1 Class Initialization

- **Configuration:** The class takes a configuration object containing settings like ResNet version, size, number of classes, initial number of filters, learning rate, weight decay, etc.
- **Network Initialization:** Initializes the ResNet model with the specified configuration.
- **Loss Function:** Cross-Entropy Loss for classification tasks.
- **Optimizer:** Stochastic Gradient Descent (SGD) with momentum and L2 weight decay.
- **Learning Rate Scheduler:** A scheduler that decays the learning rate by a factor of 0.1 every predefined number of epochs.

5.2 Training Process

- **Training Mode:** Sets the network to training mode.
- **Batch Processing:** Divides training data into batches, with each batch undergoing preprocessing, forward pass, loss calculation, and optimization.
- **Learning Rate Adjustment:** Adjusts the learning rate at the end of each epoch.
- **Checkpoint Saving:** Saves the model parameters at specified intervals.

5.3 Testing or Validation

- **Evaluation Mode:** Sets the network to evaluation mode.
- **Model Loading:** Loads the model checkpoint for validation or testing.
- **Batch Processing:** Processes each image individually for prediction and accuracy calculation.

5.4 Saving and Loading Checkpoints

- **Save:** Saves the model's state dictionary at a specified checkpoint path.
- **Load:** Loads the model's state dictionary from a checkpoint file.

5.5 Key Features

- **Device Compatibility:** Supports computation on GPU if available.
- **Dynamic Learning Rate:** Allows for dynamic adjustment of the learning rate.
- **Checkpoint Management:** Efficient management of model training and evaluation through checkpoints.

Chapter 6

ResNet Hyperparameter Tuning and Model Evaluation

This chapter outlines the process of hyperparameter tuning for a ResNet model on the CIFAR-10 dataset, followed by retraining and testing the model with the best hyperparameters.

6.1 Hyperparameter Tuning

The hyperparameter tuning was conducted over various configurations of ResNet versions, block sizes, and batch sizes. The goal was to identify the combination that yields the highest accuracy on the validation set.

6.1.1 Experimental Setup

The hyperparameters varied in the experiments were:

- **ResNet Version**: 1 and 2
- **ResNet Block Size**: 18
- **Batch Size**: 128 and 256

6.1.2 Validation Results

The model was validated with different hyperparameter combinations, and the results are as follows:

Model Version	ResNet Version	ResNet Block Size	Batch Size	Accuracy	Epoch
1	1	18	128	0.8484	30
2	1	18	256	0.8118	30
3	2	18	128	0.8848	30
4	2	18	256	0.8592	30

Table 6.1: Validation results for different hyperparameter configurations

6.1.3 Selection of Best Hyperparameters

Based on the validation results, the best performing model was Model Version 3 with the following hyperparameters:

- ResNet Version: 2
- ResNet Block Size: 18
- Batch Size: 128
- Validation Accuracy: 0.8848
- Epoch: 30

6.2 Final Model Training and Testing

With the best hyperparameters identified, the model was retrained on the full training set and then tested on the test set.

6.2.1 Testing Results

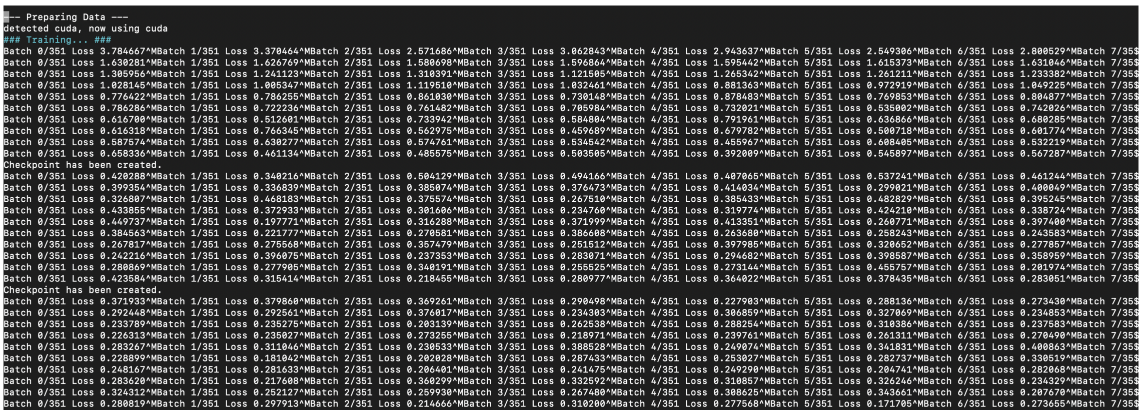
The final model achieved a test accuracy of **0.8836** on the CIFAR-10 test set.

6.3 Conclusion

The hyperparameter tuning process successfully identified the optimal configuration for the ResNet model, leading to high accuracy on the CIFAR-10 dataset. The final model, with ResNet Version 2, Size 18, and Batch Size 128, demonstrated robust performance in both validation and testing phases.

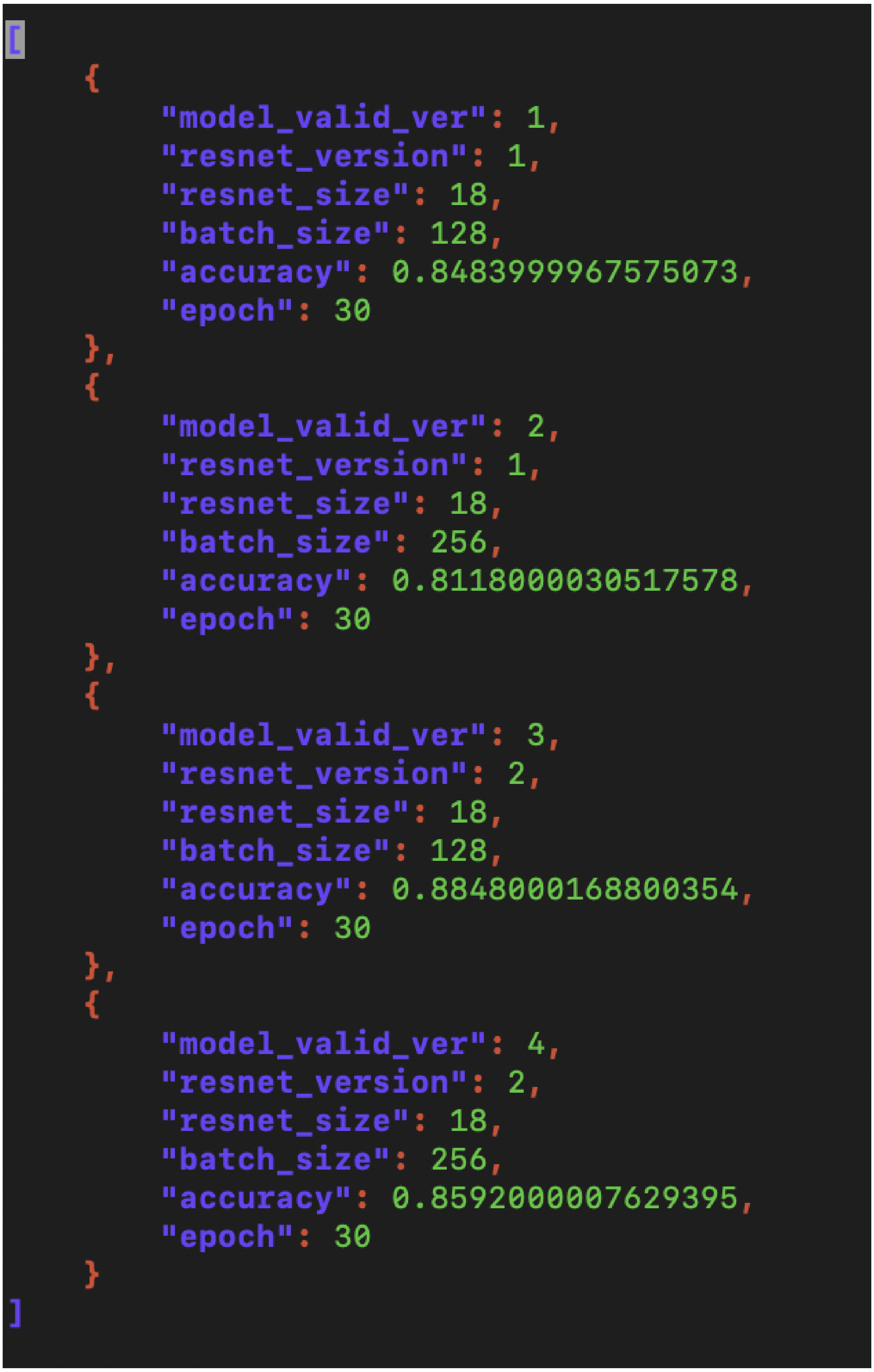
Appendix A

Hyperparameter Tuning console



```
--- Preparing Data ---
detected cuda, now using cuda
### Training... ###
Batch 0/351 Loss 3.784667^MBatch 1/351 Loss 3.370464^MBatch 2/351 Loss 2.571686^MBatch 3/351 Loss 3.062843^MBatch 4/351 Loss 2.943637^MBatch 5/351 Loss 2.549306^MBatch 6/351 Loss 2.800529^MBatch 7/35$
Batch 0/351 Loss 1.630281^MBatch 1/351 Loss 1.626769^MBatch 2/351 Loss 1.580698^MBatch 3/351 Loss 1.596864^MBatch 4/351 Loss 1.595442^MBatch 5/351 Loss 1.615373^MBatch 6/351 Loss 1.631046^MBatch 7/35$
Batch 0/351 Loss 1.305956^MBatch 1/351 Loss 1.241123^MBatch 2/351 Loss 1.310391^MBatch 3/351 Loss 1.121505^MBatch 4/351 Loss 1.265342^MBatch 5/351 Loss 1.261211^MBatch 6/351 Loss 1.233382^MBatch 7/35$
Batch 0/351 Loss 1.028145^MBatch 1/351 Loss 1.005347^MBatch 2/351 Loss 1.119510^MBatch 3/351 Loss 1.032461^MBatch 4/351 Loss 0.881363^MBatch 5/351 Loss 0.972919^MBatch 6/351 Loss 1.049225^MBatch 7/35$
Batch 0/351 Loss 0.776422^MBatch 1/351 Loss 0.786255^MBatch 2/351 Loss 0.861030^MBatch 3/351 Loss 0.730148^MBatch 4/351 Loss 0.878483^MBatch 5/351 Loss 0.769853^MBatch 6/351 Loss 0.804877^MBatch 7/35$
Batch 0/351 Loss 0.786286^MBatch 1/351 Loss 0.722236^MBatch 2/351 Loss 0.761482^MBatch 3/351 Loss 0.705984^MBatch 4/351 Loss 0.732021^MBatch 5/351 Loss 0.535002^MBatch 6/351 Loss 0.742026^MBatch 7/35$
Batch 0/351 Loss 0.616700^MBatch 1/351 Loss 0.512601^MBatch 2/351 Loss 0.733942^MBatch 3/351 Loss 0.584804^MBatch 4/351 Loss 0.791961^MBatch 5/351 Loss 0.636866^MBatch 6/351 Loss 0.680285^MBatch 7/35$
Batch 0/351 Loss 0.616318^MBatch 1/351 Loss 0.766345^MBatch 2/351 Loss 0.562975^MBatch 3/351 Loss 0.459689^MBatch 4/351 Loss 0.679782^MBatch 5/351 Loss 0.500718^MBatch 6/351 Loss 0.601774^MBatch 7/35$
Batch 0/351 Loss 0.587574^MBatch 1/351 Loss 0.630277^MBatch 2/351 Loss 0.574761^MBatch 3/351 Loss 0.534542^MBatch 4/351 Loss 0.455967^MBatch 5/351 Loss 0.608405^MBatch 6/351 Loss 0.532219^MBatch 7/35$
Batch 0/351 Loss 0.658336^MBatch 1/351 Loss 0.461134^MBatch 2/351 Loss 0.485575^MBatch 3/351 Loss 0.503505^MBatch 4/351 Loss 0.392009^MBatch 5/351 Loss 0.545897^MBatch 6/351 Loss 0.567287^MBatch 7/35$
Checkpoint has been created.
Batch 0/351 Loss 0.420288^MBatch 1/351 Loss 0.340216^MBatch 2/351 Loss 0.504129^MBatch 3/351 Loss 0.494166^MBatch 4/351 Loss 0.407065^MBatch 5/351 Loss 0.537241^MBatch 6/351 Loss 0.461244^MBatch 7/35$
Batch 0/351 Loss 0.399354^MBatch 1/351 Loss 0.336839^MBatch 2/351 Loss 0.385074^MBatch 3/351 Loss 0.376473^MBatch 4/351 Loss 0.414034^MBatch 5/351 Loss 0.299021^MBatch 6/351 Loss 0.400049^MBatch 7/35$
Batch 0/351 Loss 0.326807^MBatch 1/351 Loss 0.468183^MBatch 2/351 Loss 0.375574^MBatch 3/351 Loss 0.267510^MBatch 4/351 Loss 0.385433^MBatch 5/351 Loss 0.482829^MBatch 6/351 Loss 0.395245^MBatch 7/35$
Batch 0/351 Loss 0.433855^MBatch 1/351 Loss 0.372933^MBatch 2/351 Loss 0.301606^MBatch 3/351 Loss 0.234760^MBatch 4/351 Loss 0.319774^MBatch 5/351 Loss 0.424210^MBatch 6/351 Loss 0.338724^MBatch 7/35$
Batch 0/351 Loss 0.449737^MBatch 1/351 Loss 0.197771^MBatch 2/351 Loss 0.316288^MBatch 3/351 Loss 0.371999^MBatch 4/351 Loss 0.413351^MBatch 5/351 Loss 0.260771^MBatch 6/351 Loss 0.397400^MBatch 7/35$
Batch 0/351 Loss 0.384563^MBatch 1/351 Loss 0.221777^MBatch 2/351 Loss 0.270581^MBatch 3/351 Loss 0.386608^MBatch 4/351 Loss 0.263680^MBatch 5/351 Loss 0.258243^MBatch 6/351 Loss 0.243583^MBatch 7/35$
Batch 0/351 Loss 0.267817^MBatch 1/351 Loss 0.275568^MBatch 2/351 Loss 0.357479^MBatch 3/351 Loss 0.251512^MBatch 4/351 Loss 0.397985^MBatch 5/351 Loss 0.320652^MBatch 6/351 Loss 0.277857^MBatch 7/35$
Batch 0/351 Loss 0.242216^MBatch 1/351 Loss 0.396075^MBatch 2/351 Loss 0.237353^MBatch 3/351 Loss 0.283071^MBatch 4/351 Loss 0.294682^MBatch 5/351 Loss 0.398587^MBatch 6/351 Loss 0.358959^MBatch 7/35$
Batch 0/351 Loss 0.280869^MBatch 1/351 Loss 0.277905^MBatch 2/351 Loss 0.340191^MBatch 3/351 Loss 0.255525^MBatch 4/351 Loss 0.273144^MBatch 5/351 Loss 0.455757^MBatch 6/351 Loss 0.201974^MBatch 7/35$
Batch 0/351 Loss 0.423584^MBatch 1/351 Loss 0.315414^MBatch 2/351 Loss 0.218455^MBatch 3/351 Loss 0.280977^MBatch 4/351 Loss 0.364022^MBatch 5/351 Loss 0.378435^MBatch 6/351 Loss 0.283051^MBatch 7/35$
Checkpoint has been created.
Batch 0/351 Loss 0.371933^MBatch 1/351 Loss 0.379860^MBatch 2/351 Loss 0.369261^MBatch 3/351 Loss 0.290498^MBatch 4/351 Loss 0.227903^MBatch 5/351 Loss 0.288136^MBatch 6/351 Loss 0.273430^MBatch 7/35$
Batch 0/351 Loss 0.292448^MBatch 1/351 Loss 0.292561^MBatch 2/351 Loss 0.376017^MBatch 3/351 Loss 0.234303^MBatch 4/351 Loss 0.306859^MBatch 5/351 Loss 0.327069^MBatch 6/351 Loss 0.234853^MBatch 7/35$
Batch 0/351 Loss 0.233789^MBatch 1/351 Loss 0.235275^MBatch 2/351 Loss 0.203139^MBatch 3/351 Loss 0.262538^MBatch 4/351 Loss 0.288254^MBatch 5/351 Loss 0.310386^MBatch 6/351 Loss 0.237583^MBatch 7/35$
Batch 0/351 Loss 0.226313^MBatch 1/351 Loss 0.235027^MBatch 2/351 Loss 0.273255^MBatch 3/351 Loss 0.218971^MBatch 4/351 Loss 0.239761^MBatch 5/351 Loss 0.261311^MBatch 6/351 Loss 0.270490^MBatch 7/35$
Batch 0/351 Loss 0.283267^MBatch 1/351 Loss 0.311046^MBatch 2/351 Loss 0.230533^MBatch 3/351 Loss 0.388528^MBatch 4/351 Loss 0.249074^MBatch 5/351 Loss 0.341831^MBatch 6/351 Loss 0.400863^MBatch 7/35$
Batch 0/351 Loss 0.228899^MBatch 1/351 Loss 0.181042^MBatch 2/351 Loss 0.202028^MBatch 3/351 Loss 0.287433^MBatch 4/351 Loss 0.253027^MBatch 5/351 Loss 0.282737^MBatch 6/351 Loss 0.330519^MBatch 7/35$
Batch 0/351 Loss 0.248167^MBatch 1/351 Loss 0.281633^MBatch 2/351 Loss 0.206401^MBatch 3/351 Loss 0.241475^MBatch 4/351 Loss 0.249290^MBatch 5/351 Loss 0.204741^MBatch 6/351 Loss 0.282068^MBatch 7/35$
Batch 0/351 Loss 0.283620^MBatch 1/351 Loss 0.217608^MBatch 2/351 Loss 0.360299^MBatch 3/351 Loss 0.332592^MBatch 4/351 Loss 0.310857^MBatch 5/351 Loss 0.326246^MBatch 6/351 Loss 0.234329^MBatch 7/35$
Batch 0/351 Loss 0.324312^MBatch 1/351 Loss 0.252127^MBatch 2/351 Loss 0.259930^MBatch 3/351 Loss 0.267480^MBatch 4/351 Loss 0.308625^MBatch 5/351 Loss 0.343661^MBatch 6/351 Loss 0.207670^MBatch 7/35$
Batch 0/351 Loss 0.280819^MBatch 1/351 Loss 0.297913^MBatch 2/351 Loss 0.214666^MBatch 3/351 Loss 0.310200^MBatch 4/351 Loss 0.277568^MBatch 5/351 Loss 0.171705^MBatch 6/351 Loss 0.273655^MBatch 7/35$
```

Figure A.1: `Tuning console log`

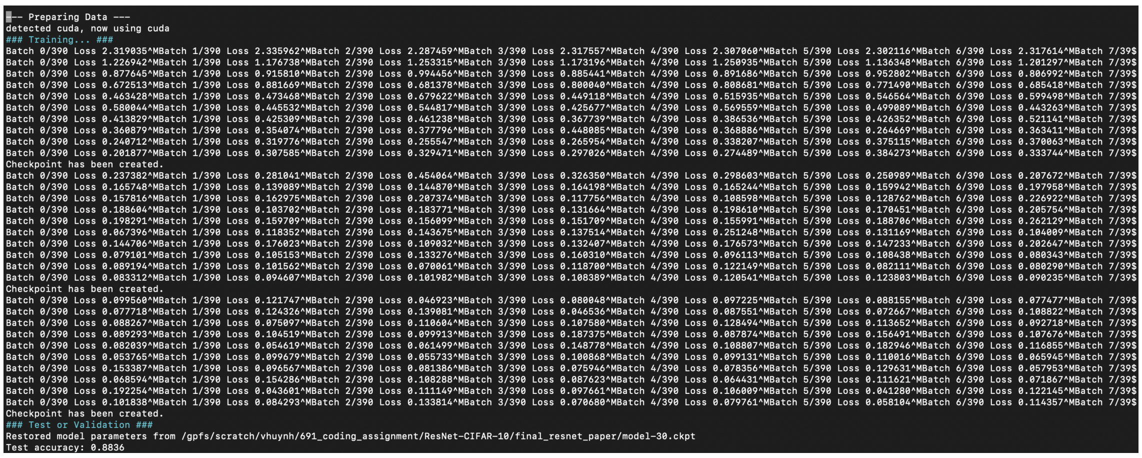


```
[
    {
        "model_valid_ver": 1,
        "resnet_version": 1,
        "resnet_size": 18,
        "batch_size": 128,
        "accuracy": 0.8483999967575073,
        "epoch": 30
    },
    {
        "model_valid_ver": 2,
        "resnet_version": 1,
        "resnet_size": 18,
        "batch_size": 256,
        "accuracy": 0.8118000030517578,
        "epoch": 30
    },
    {
        "model_valid_ver": 3,
        "resnet_version": 2,
        "resnet_size": 18,
        "batch_size": 128,
        "accuracy": 0.8848000168800354,
        "epoch": 30
    },
    {
        "model_valid_ver": 4,
        "resnet_version": 2,
        "resnet_size": 18,
        "batch_size": 256,
        "accuracy": 0.8592000007629395,
        "epoch": 30
    }
]
```

Figure A.2: `Hyper-paramters log`

Appendix B

Testing console



```
--- Preparing Data ---
detected cuda, now using cuda
### Training... ###
Batch 0/390 Loss 2.319035^MBatch 1/390 Loss 2.335962^MBatch 2/390 Loss 2.287459^MBatch 3/390 Loss 2.317557^MBatch 4/390 Loss 2.307060^MBatch 5/390 Loss 2.302116^MBatch 6/390 Loss 2.317614^MBatch 7/39$
Batch 0/390 Loss 1.226942^MBatch 1/390 Loss 1.176738^MBatch 2/390 Loss 1.253315^MBatch 3/390 Loss 1.173196^MBatch 4/390 Loss 1.250935^MBatch 5/390 Loss 1.136348^MBatch 6/390 Loss 1.201297^MBatch 7/39$
Batch 0/390 Loss 0.877645^MBatch 1/390 Loss 0.915810^MBatch 2/390 Loss 0.994456^MBatch 3/390 Loss 0.885441^MBatch 4/390 Loss 0.891686^MBatch 5/390 Loss 0.952802^MBatch 6/390 Loss 0.806992^MBatch 7/39$
Batch 0/390 Loss 0.672513^MBatch 1/390 Loss 0.881669^MBatch 2/390 Loss 0.681378^MBatch 3/390 Loss 0.800040^MBatch 4/390 Loss 0.808681^MBatch 5/390 Loss 0.771490^MBatch 6/390 Loss 0.685418^MBatch 7/39$
Batch 0/390 Loss 0.463428^MBatch 1/390 Loss 0.473468^MBatch 2/390 Loss 0.679622^MBatch 3/390 Loss 0.449118^MBatch 4/390 Loss 0.515935^MBatch 5/390 Loss 0.546564^MBatch 6/390 Loss 0.599498^MBatch 7/39$
Batch 0/390 Loss 0.580044^MBatch 1/390 Loss 0.445532^MBatch 2/390 Loss 0.544817^MBatch 3/390 Loss 0.425677^MBatch 4/390 Loss 0.569559^MBatch 5/390 Loss 0.499089^MBatch 6/390 Loss 0.443263^MBatch 7/39$
Batch 0/390 Loss 0.413829^MBatch 1/390 Loss 0.425309^MBatch 2/390 Loss 0.461238^MBatch 3/390 Loss 0.367739^MBatch 4/390 Loss 0.386536^MBatch 5/390 Loss 0.426352^MBatch 6/390 Loss 0.521141^MBatch 7/39$
Batch 0/390 Loss 0.360879^MBatch 1/390 Loss 0.354074^MBatch 2/390 Loss 0.377796^MBatch 3/390 Loss 0.448085^MBatch 4/390 Loss 0.368886^MBatch 5/390 Loss 0.264669^MBatch 6/390 Loss 0.363411^MBatch 7/39$
Batch 0/390 Loss 0.240712^MBatch 1/390 Loss 0.319776^MBatch 2/390 Loss 0.255547^MBatch 3/390 Loss 0.265954^MBatch 4/390 Loss 0.338207^MBatch 5/390 Loss 0.375115^MBatch 6/390 Loss 0.370063^MBatch 7/39$
Batch 0/390 Loss 0.201877^MBatch 1/390 Loss 0.307585^MBatch 2/390 Loss 0.329471^MBatch 3/390 Loss 0.297026^MBatch 4/390 Loss 0.274489^MBatch 5/390 Loss 0.384273^MBatch 6/390 Loss 0.333744^MBatch 7/39$
Checkpoint has been created.
Batch 0/390 Loss 0.237382^MBatch 1/390 Loss 0.281041^MBatch 2/390 Loss 0.454064^MBatch 3/390 Loss 0.326350^MBatch 4/390 Loss 0.298603^MBatch 5/390 Loss 0.250989^MBatch 6/390 Loss 0.207672^MBatch 7/39$
Batch 0/390 Loss 0.165748^MBatch 1/390 Loss 0.139089^MBatch 2/390 Loss 0.144870^MBatch 3/390 Loss 0.164198^MBatch 4/390 Loss 0.165244^MBatch 5/390 Loss 0.159942^MBatch 6/390 Loss 0.197958^MBatch 7/39$
Batch 0/390 Loss 0.157816^MBatch 1/390 Loss 0.162975^MBatch 2/390 Loss 0.207374^MBatch 3/390 Loss 0.117756^MBatch 4/390 Loss 0.108598^MBatch 5/390 Loss 0.128762^MBatch 6/390 Loss 0.226922^MBatch 7/39$
Batch 0/390 Loss 0.188604^MBatch 1/390 Loss 0.103702^MBatch 2/390 Loss 0.183771^MBatch 3/390 Loss 0.131664^MBatch 4/390 Loss 0.198610^MBatch 5/390 Loss 0.170451^MBatch 6/390 Loss 0.205754^MBatch 7/39$
Batch 0/390 Loss 0.198291^MBatch 1/390 Loss 0.159709^MBatch 2/390 Loss 0.156099^MBatch 3/390 Loss 0.151709^MBatch 4/390 Loss 0.155991^MBatch 5/390 Loss 0.188706^MBatch 6/390 Loss 0.262129^MBatch 7/39$
Batch 0/390 Loss 0.067396^MBatch 1/390 Loss 0.118352^MBatch 2/390 Loss 0.143675^MBatch 3/390 Loss 0.137514^MBatch 4/390 Loss 0.251248^MBatch 5/390 Loss 0.131169^MBatch 6/390 Loss 0.104009^MBatch 7/39$
Batch 0/390 Loss 0.144706^MBatch 1/390 Loss 0.176023^MBatch 2/390 Loss 0.109032^MBatch 3/390 Loss 0.132407^MBatch 4/390 Loss 0.176573^MBatch 5/390 Loss 0.147233^MBatch 6/390 Loss 0.202647^MBatch 7/39$
Batch 0/390 Loss 0.079101^MBatch 1/390 Loss 0.105153^MBatch 2/390 Loss 0.133276^MBatch 3/390 Loss 0.160310^MBatch 4/390 Loss 0.096113^MBatch 5/390 Loss 0.108438^MBatch 6/390 Loss 0.080343^MBatch 7/39$
Batch 0/390 Loss 0.089194^MBatch 1/390 Loss 0.101562^MBatch 2/390 Loss 0.070061^MBatch 3/390 Loss 0.118700^MBatch 4/390 Loss 0.122149^MBatch 5/390 Loss 0.082111^MBatch 6/390 Loss 0.080290^MBatch 7/39$
Batch 0/390 Loss 0.083312^MBatch 1/390 Loss 0.094607^MBatch 2/390 Loss 0.101982^MBatch 3/390 Loss 0.108389^MBatch 4/390 Loss 0.120541^MBatch 5/390 Loss 0.123803^MBatch 6/390 Loss 0.090235^MBatch 7/39$
Checkpoint has been created.
Batch 0/390 Loss 0.099560^MBatch 1/390 Loss 0.121747^MBatch 2/390 Loss 0.046923^MBatch 3/390 Loss 0.080048^MBatch 4/390 Loss 0.097225^MBatch 5/390 Loss 0.088155^MBatch 6/390 Loss 0.077477^MBatch 7/39$
Batch 0/390 Loss 0.077718^MBatch 1/390 Loss 0.124326^MBatch 2/390 Loss 0.139081^MBatch 3/390 Loss 0.046536^MBatch 4/390 Loss 0.087551^MBatch 5/390 Loss 0.072667^MBatch 6/390 Loss 0.108822^MBatch 7/39$
Batch 0/390 Loss 0.088267^MBatch 1/390 Loss 0.075097^MBatch 2/390 Loss 0.110604^MBatch 3/390 Loss 0.107580^MBatch 4/390 Loss 0.128494^MBatch 5/390 Loss 0.113652^MBatch 6/390 Loss 0.092718^MBatch 7/39$
Batch 0/390 Loss 0.089293^MBatch 1/390 Loss 0.104519^MBatch 2/390 Loss 0.099913^MBatch 3/390 Loss 0.187375^MBatch 4/390 Loss 0.087874^MBatch 5/390 Loss 0.156491^MBatch 6/390 Loss 0.107676^MBatch 7/39$
Batch 0/390 Loss 0.082039^MBatch 1/390 Loss 0.054619^MBatch 2/390 Loss 0.061499^MBatch 3/390 Loss 0.148778^MBatch 4/390 Loss 0.108807^MBatch 5/390 Loss 0.182946^MBatch 6/390 Loss 0.116855^MBatch 7/39$
Batch 0/390 Loss 0.053765^MBatch 1/390 Loss 0.099679^MBatch 2/390 Loss 0.055733^MBatch 3/390 Loss 0.100868^MBatch 4/390 Loss 0.099131^MBatch 5/390 Loss 0.110016^MBatch 6/390 Loss 0.065945^MBatch 7/39$
Batch 0/390 Loss 0.153387^MBatch 1/390 Loss 0.096567^MBatch 2/390 Loss 0.081386^MBatch 3/390 Loss 0.075946^MBatch 4/390 Loss 0.078356^MBatch 5/390 Loss 0.129631^MBatch 6/390 Loss 0.057953^MBatch 7/39$
Batch 0/390 Loss 0.068594^MBatch 1/390 Loss 0.154286^MBatch 2/390 Loss 0.108288^MBatch 3/390 Loss 0.087623^MBatch 4/390 Loss 0.064431^MBatch 5/390 Loss 0.111621^MBatch 6/390 Loss 0.071867^MBatch 7/39$
Batch 0/390 Loss 0.192254^MBatch 1/390 Loss 0.043601^MBatch 2/390 Loss 0.111149^MBatch 3/390 Loss 0.097661^MBatch 4/390 Loss 0.106009^MBatch 5/390 Loss 0.041280^MBatch 6/390 Loss 0.122145^MBatch 7/39$
Batch 0/390 Loss 0.101838^MBatch 1/390 Loss 0.084293^MBatch 2/390 Loss 0.133814^MBatch 3/390 Loss 0.070680^MBatch 4/390 Loss 0.079761^MBatch 5/390 Loss 0.058104^MBatch 6/390 Loss 0.114357^MBatch 7/39$
Checkpoint has been created.
### Test or Validation ###
Restored model parameters from /gpfs/scratch/vhuynh/691_coding_assignment/ResNet-CIFAR-10/final_resnet_paper/model-30.ckpt
Test accuracy: 0.8836
```

Figure B.1: `Hyper-paramters log`