ChatGPT

Designing ResNet-Based Model m8 for CIFAR-10

To meet the 91%+ accuracy target under 400k parameters, we adopt a **ResNet-20** style architecture with strict adherence to the original ResNet paper's CIFAR-10 methodology. The model uses 3×3 convolutional layers in residual blocks, batch normalization after each convolution (before ReLU), and identity shortcuts with **projection only when downsampling** (ResNet "option B") for efficiency [1]. We **do not use dropout**, and we apply the exact data preprocessing and augmentation described by He et al. (2015): **per-channel standardization** (zero-mean, unit-variance) and CIFAR-10's simple augmentation (4-pixel zero-padding with random 32×32 cropping and horizontal flip).

Below are the updates to implement **model m8**, along with the new data loader and configuration:

Updated `model.py` – Implementing `build_model_m8()`

The `build_model_m8()` function constructs a ResNet-20-like network for CIFAR-10. It starts with an initial 3×3 conv layer (16 filters), followed by three stages of residual blocks. Each stage has 3 blocks of two 3×3 conv layers: - **Stage 1:** 3 blocks with 16 filters (no downsampling in this stage). - **Stage 2:** 3 blocks with 32 filters; the first block uses stride 2 and a 1×1 conv shortcut to downsample from 16→32 channels. - **Stage 3:** 3 blocks with 64 filters; the first block uses stride 2 and a 1×1 conv shortcut to downsample 32→64 channels.

All shortcut connections for **dimension increases** use a 1×1 conv (stride 2) to match the new shape (as recommended by ResNet option B, which yields slightly better accuracy than zero-padding) [1]. In stages where the input/output dimensions remain the same, we use identity shortcuts with no extra parameters. After the final residual block, we apply global average pooling and a 10-class fully-connected output. No dropout is applied in this architecture, aligning with the ResNet methodology (no dropout since batch normalization is used). We use an L2 weight regularizer (`reg`) with λ=1e-4 on conv and dense layers.

```
def build_model_m8(config):
    """
    Builds the ResNet-20-like model for CIFAR-10 (model m8), following the
ResNet paper.
    Uses 3x3 conv layers, batch normalization, ReLU, and residual shortcuts.
    No dropout is used (dropout disabled for ResNet).
    """
    import tensorflow as tf
    from keras.api.models import Model
    from keras.api.layers import Input, Conv2D, BatchNormalization, Activation
    from keras.api.layers import Add, GlobalAveragePooling2D, Dense

    # L2 weight decay regularizer (if enabled in config)
    reg = tf.keras.regularizers.l2(config.L2_MODE["lambda"]) if
```

```
config.L2_MODE["enabled"] else None

    # Input layer for 32x32 RGB images
    input_layer = Input(shape=(32, 32, 3))
    x = Conv2D(16, (3, 3), padding="same", kernel_regularizer=reg)(input_layer)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    # Define a residual block helper function
    def res_block(x, filters, downsample=False):
        """
        Defines a 2-layer residual block with optional downsampling.
        If downsample=True, applies stride-2 convolution and projection
shortcut.
        """
        stride = 2 if downsample else 1
        shortcut = x  # Save input tensor for shortcut
        # First conv layer (with stride if downsampling)
        y = Conv2D(filters, (3, 3), strides=stride, padding="same",
kernel_regularizer=reg)(x)
        y = BatchNormalization()(y)
        y = Activation("relu")(y)
        # Second conv layer
        y = Conv2D(filters, (3, 3), padding="same", kernel_regularizer=reg)(y)
        y = BatchNormalization()(y)
        # Projection shortcut if downsampling or channel mismatch
        if downsample or shortcut.shape[-1] != filters:
            shortcut = Conv2D(filters, (1, 1), strides=stride, padding="same",
                              kernel_regularizer=reg)(shortcut)
        # Add shortcut and apply ReLU
        out = Add()([y, shortcut])
        out = Activation("relu")(out)
        return out

    # Stage 1: 3 residual blocks with 16 filters (no downsampling in this stage)
    for _ in range(3):
        x = res_block(x, filters=16)

    # Stage 2: 3 residual blocks with 32 filters (first block downsamples)
    x = res_block(x, filters=32, downsample=True)
    for _ in range(2):
        x = res_block(x, filters=32)

    # Stage 3: 3 residual blocks with 64 filters (first block downsamples)
    x = res_block(x, filters=64, downsample=True)
    for _ in range(2):
        x = res_block(x, filters=64)
```

```
    # Global Average Pooling and output layer
    x = GlobalAveragePooling2D()(x)
    prediction_layer = Dense(10, activation="softmax", kernel_regularizer=reg)
(x)

    # Create and compile the model
    model = Model(inputs=input_layer, outputs=prediction_layer)
    optimizer = tf.keras.optimizers.SGD(learning_rate=0.1, momentum=0.9,
nesterov=False)
    model.compile(optimizer=optimizer,
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                  metrics=["accuracy"])
    model.summary()  # Print model architecture summary
    return model
```

To integrate this in a modular way, we introduce a dispatcher function `dispatch_build_model`. This mirrors the dataset loader dispatch mechanism and dynamically calls the correct builder based on the model number:

```
def dispatch_build_model(model_number, config):
    """
    Routes to the appropriate model-building function based on model_number.
    For example, model_number=8 will call build_model_m8().
    """
    print("\n   dispatch_build_model")
    try:
        build_fn = globals()[f"build_model_m{model_number}"]
        return build_fn(config)
    except KeyError:
        raise ValueError(f"  ValueError: build_model_m{model_number}() is not
defined for model_number={model_number}")
```

Now `dispatch_build_model(8, config)` will invoke `build_model_m8(config)`. The new Model m8 has ~0.27M parameters (well under 400k) and is expected to reach **≈91–92%** test accuracy in one run, consistent with ResNet-20 results.

Updated `data.py` – CIFAR-10 Loader `load_dataset_m8()`

We add a corresponding data loader for model m8 that applies the **exact preprocessing and augmentation** from the ResNet paper. This loader performs per-channel normalization and the standard pad-and-crop augmentation:

- **Standardization:** Convert images to float32 `[0,1]`, then subtract the CIFAR-10 mean and divide by std for each channel (using pre-computed dataset statistics).

- **Augmentation:** For training data, apply a 4-pixel zero-padding on each side, then take a random 32×32 crop, and randomly flip horizontally. *(No color jitter or extra augmentations are used, to strictly follow the original setup.)*
- For testing, no augmentation is applied beyond normalization; we evaluate the original 32×32 image.

The `_load_dataset_m8()` function returns the processed training and test sets (with the last 5,000 training images held out if following the 45k/5k train/val split as in the paper):

```
# CIFAR-10 mean/std (for normalization)
_CIFAR10_MEAN = [0.4914, 0.4822, 0.4465]
_CIFAR10_STD  = [0.2023, 0.1994, 0.2010]

def _load_dataset_m8(config):
    """
    Loads and preprocesses CIFAR-10 for model m8.
    Applies per-channel standardization and ResNet paper's data augmentation
    (4-pixel padding + random crop + horizontal flip) for training images.
    """
    from torchvision import datasets, transforms
    import numpy as np

    print("\n   _load_dataset_m8")
    # Load CIFAR-10 dataset
    train_set = datasets.CIFAR10(root=config.DATA_PATH, train=True,
download=True)
    test_set = datasets.CIFAR10(root=config.DATA_PATH, train=False,
download=True)

    train_images = train_set.data  # uint8 array of shape (50000, 32, 32, 3)
    test_images = test_set.data    # uint8 array of shape (10000, 32, 32, 3)
    train_labels = np.array(train_set.targets)
    test_labels = np.array(test_set.targets)

    # If not in LIGHT_MODE, reserve last 5k training images as validation (45k
train as in ResNet paper)
    if config.LIGHT_MODE:
        train_images = train_images[:1000];  train_labels = train_labels[:1000]
        test_images = test_images[:200];     test_labels = test_labels[:200]
    else:
        train_images = train_images[:-5000]; train_labels = train_labels[:-5000]
        # (Note: the 5k set aside can be used for validation if needed)

    # Define augmentation pipeline for training images (pad, crop, flip), then
normalize
    if config.AUGMENT_MODE:
        transform = transforms.Compose([
```

```
            transforms.ToPILImage(),
            transforms.RandomCrop(32, padding=4),      # pad 4 pixels on each
side and crop
            transforms.RandomHorizontalFlip(),         # random horizontal flip
            transforms.ToTensor(),                    # convert to tensor
[C,H,W] in [0.0,1.0]
            transforms.Normalize(mean=_CIFAR10_MEAN, std=_CIFAR10_STD)  #
channel-wise standardization
        ])
        # Apply transforms to each training image (returns torch.Tensor per
image)
        augmented_tensors = [transform(img) for img in train_images]
        # Convert list of tensors to numpy array in shape (N, 32, 32, 3)
        train_data = np.stack([tensor.permute(1, 2, 0).numpy() for tensor in
augmented_tensors], axis=0)
    else:
        # If no augmentation, just standardize the training images directly
        train_data = train_images.astype(np.float32) / 255.0
        for i in range(3):
            train_data[..., i] = (train_data[..., i] - _CIFAR10_MEAN[i]) /
_CIFAR10_STD[i]

    # Standardize the test set (no augmentation)
    test_data = test_images.astype(np.float32) / 255.0
    for i in range(3):
        test_data[..., i] = (test_data[..., i] - _CIFAR10_MEAN[i]) /
_CIFAR10_STD[i]

    return train_data, train_labels, test_data, test_labels
```

This loader ensures the model sees the same data preprocessing that yielded success in ResNet: **normalized inputs and classic CIFAR-10 augmentation**. The function can be invoked via the existing dispatcher (i.e., `dispatch_load_dataset(8, config)` will call `_load_dataset_m8`).

New `m8_base.json` – Training Configuration

The configuration file for m8 (`m8_base.json`) specifies the training hyperparameters exactly as in the ResNet paper, and fixes issues from previous models (m7) by using the proven ResNet settings that made m6 successful:

- **Optimizer:** Stochastic Gradient Descent (SGD) with 0.9 momentum, initial learning rate **0.1**.
- **Weight Decay:** L2 regularization with λ = **0.0001** on weights (note: this is lower than m6/m7's 0.0005, matching the ResNet paper).
- **Batch Size: 128** (larger batch as used in the paper for stable training).

- **Learning Rate Schedule:** Manual step decay – the learning rate will be multiplied by 0.1 at **32k and 48k iterations** (approximately at epochs 90 and 135 out of 200). We train for a total of **64k iterations (~200 epochs)** to fully converge as in the original setup.
- **No Early Stopping:** We disable early stopping so that training runs the full 64k iterations, ensuring the model benefits from later learning rate drops. Instead, one can monitor validation accuracy on the 5k held-out set and take the final model or best model.
- **Data Augmentation:** Enabled (true) to apply the pad/crop/flip on training images.
- **Dropout:** Disabled (false) since ResNet did not use dropout.
- Other paths and settings remain consistent with prior models.

```
{
  "CONFIG_PATH": "artifact/config",
  "DATA_PATH": "artifact/data",
  "LOG_PATH": "artifact/log",
  "CHECKPOINT_PATH": "artifact/checkpoint",
  "RESULT_PATH": "artifact/result",
  "MODEL_PATH": "artifact/model/",
  "ERROR_PATH": "artifact/error",

  "LIGHT_MODE": false,
  "AUGMENT_MODE": true,

  "L2_MODE": {
    "enabled": true,
    "lambda": 0.0001
  },
  "DROPOUT_MODE": {
    "enabled": false,
    "rate": 0.3
  },

  "OPTIMIZER": {
    "type": "sgd",
    "learning_rate": 0.1,
    "momentum": 0.9
  },
  "SCHEDULE_MODE": {
    "enabled": false
  },
  "EARLY_STOP_MODE": {
    "enabled": false,
    "monitor": "val_accuracy",
    "patience": 10,
    "restore_best_weights": true,
    "verbose": 1
  },
```

```
  "EPOCHS_COUNT": 200,
  "BATCH_SIZE": 128
}
```

Notes: The learning rate decay at 32k/48k iterations will be implemented via a step schedule (e.g., using a scheduler callback or manual adjustment) rather than the reduce-on-plateau used in m6/m7. The `SCHEDULE_MODE` is turned off in the config to avoid the previous plateau-based schedule – instead we follow the fixed schedule from the paper. By training for 200 epochs with these settings, Model m8 should reliably exceed **91% test accuracy** in one run, matching the reported ResNet-20 performance on CIFAR-10 while using fewer than 400k parameters.

---

[1] task-001_1512.03385v1.pdf
file://file-PCnp2X5rJVsz8EVxRn4dDZ