ChatGPT

Improving CIFAR-10 VGG16-Style Classifier: Recommendations

Architecture Improvements (Model Design)

1. **Add Batch Normalization layers:** Insert BatchNorm after each convolution layer. This stabilizes and accelerates training by normalizing activations, allowing the use of higher learning rates and deeper networks without vanishing/exploding gradients [1]. BatchNorm has become standard in modern CNNs (e.g. MobileNet applies BN after every conv [2]) and often improves generalization by reducing internal covariate shift.

2. **Replace the Flatten+Dense with Global Average Pooling:** Remove the 4096-unit Flatten→Dense(128) block and use a Global Average Pooling layer before the final 10-class output. This drastically cuts the parameter count (the ~524K parameters in the Dense layer would be eliminated, bringing the model well under 400K). Global pooling forces the network to condense spatial information into global features, which reduces overfitting and model size [2]. *For example, MobileNet uses a final average pooling to shrink the feature map to 1×1, and notes that nearly all additional parameters come from large fully-connected layers* [3]. By relying on a global pooled feature vector, we retain predictive power while **reducing parameters** dramatically.

3. **Increase Network Depth (more conv layers/blocks):** With the saved parameter budget, consider adding another convolutional block (e.g. a third block with ~128 filters followed by pooling) to make the network deeper. Deeper models can learn more complex features and generally yield higher accuracy [4]. Using multiple small 3×3 conv layers (VGG-style) is effective for increasing depth without huge parameter growth. By keeping each layer "thin" (moderate number of filters) and using an extra pooling stage, the total parameters stay within 400k. This leverages the finding that pushing depth to 16–19 layers (with small filters) gave significant accuracy improvements in VGG [4] – even a modest increase in depth for our model can improve feature extraction and generalization.

4. **Use Depthwise Separable Convolutions:** Introduce depthwise separable conv layers (from MobileNet) to replace standard conv layers for large parameter savings. A depthwise separable convolution splits a normal conv into a lightweight depthwise filter (per-channel filtering) followed by a 1×1 pointwise conv for combining channels. This can reduce the computation and weight count **by about 8–9×** for the same output dimensionality [5]. *MobileNet's design shows that such factorized conv layers achieve almost the same accuracy at a fraction of the cost* [5]. By using depthwise separable convs, we could substantially lower the param count of each conv block, potentially allowing more filters or additional layers to be added to boost accuracy while still staying under 400K total parameters.

5. **Incorporate Residual "Skip" Connections:** Re-architect the model into a **ResNet-like** network by adding skip connections that sum the output of earlier layers with later layers. These identity shortcuts introduce *no extra parameters* (an identity skip adds outputs directly [6]), yet they enable

much deeper architectures to train effectively [7]. Residual connections help preserve gradients in very deep networks, addressing optimization difficulties. This means we could stack more conv layers (increasing depth for better accuracy) without increasing model size or risking vanishing gradients. In fact, He et al. demonstrated that a 110-layer ResNet on CIFAR-10 (with many skip connections) achieved **state-of-the-art** accuracy with fewer parameters than prior thinner networks [8]. Adding residual connections would thus improve generalization by allowing a deeper model within the 400k parameter budget.

Training Strategy Improvements

1. **Apply Data Augmentation:** Use on-the-fly image augmentations to expand the effective training data. For CIFAR-10, common augmentations include random horizontal flips, small random crops/patches from images (after padding), and even slight color jitter or rotations. Augmentation helps prevent overfitting and improves generalization by exposing the model to varied image transformations. In practice, **all** high-performing models on CIFAR-10 use data augmentation – e.g. random cropping and flipping were standard in VGG/ResNet training [9]. By enabling augmentations via Keras `ImageDataGenerator` or `tf.image` (for example), the model will learn more robust features and likely achieve higher test accuracy.

2. **Implement Learning-Rate Scheduling:** Rather than a fixed learning rate, use a schedule or callback to adjust the learning rate during training. For instance, you can start with a relatively high LR and then reduce it on a schedule (step decay) or when validation accuracy plateaus. This strategy leads to better convergence: the VGG network, for example, began with an initial rate and then **decreased the LR by a factor of 10** each time validation stopped improving (doing this 3 times over ~74 epochs) [10]. Similarly, ResNet on CIFAR-10 used an initial LR of 0.1 and dropped it by 10× at specific milestones [11]. In our case, using a Keras callback like **ReduceLROnPlateau** (to lower LR when val accuracy stagnates) or manually halving the LR every few epochs can help the model escape plateaus and improve its final accuracy.

3. **Train for More Epochs (with Early Stopping if needed):** Allow the training to run for more epochs (up to the 32 epochs limit, or as needed) so the model can fully converge. The current setup of 16 epochs may be cutting training short – many CNNs require dozens of epochs to reach peak performance. For example, the VGG models were trained for **74 epochs** to get top results on ImageNet [10]. With a learning rate schedule in place, additional epochs will let the model refine its learning. To avoid overfitting in these extra epochs, use an **EarlyStopping** callback monitoring validation loss/accuracy (or rely on the saved best model checkpoint) to stop training once the validation metrics stop improving. This ensures we take advantage of the allowable epochs for better accuracy, while not running too long or overfitting once the model has converged.

4. **Use SGD + Momentum Optimizer:** Consider switching from Adam to **SGD with momentum (0.9)** for training, combined with the above LR schedule. Gradient descent with momentum is the optimizer used in most seminal CNN results – both VGG and ResNet were trained with momentum SGD (with momentum=0.9) and a proper LR decay schedule [11] [10]. In many vision tasks, SGD tends to yield better final generalization than Adam, given enough training time and tuning. If using SGD, start with an appropriate learning rate (e.g. 0.05–0.1 for batch-normalized networks [11]) and decay it as discussed. You may also increase the batch size (e.g. 32 or 64 instead of 16) to stabilize updates when using SGD. Overall, this change can lead to a small but meaningful boost in test accuracy and

more stable training, at the cost of potentially needing a few more epochs to converge compared to Adam.

Regularization Techniques

1. **Apply Weight Decay (L2 Regularization):** Add a weight decay (L2 penalty) to the network's weights (especially conv and dense kernels) to discourage overfitting. This can be done by setting a `kernel_regularizer=tf.keras.regularizers.L2(lambda)` on layers or via the optimizer's decay parameter. Weight decay was crucial in training VGG-style networks – they used a penalty multiplier of 5×10^−4 on weights [10] – and ResNet likewise used 1×10^−4 in CIFAR experiments [12]. The L2 regularization will gently constrain the magnitude of weights, acting as a regularizer that improves the model's ability to generalize to the test set (especially important given the relatively small dataset).

2. **Add Dropout Layers:** Introduce dropout to the model to randomly zero-out a fraction of activations during training, which helps combat overfitting. For example, you could add a `Dropout(0.5)` layer after the final convolution block (before the classifier), or if any dense layer remains, apply dropout there (VGG applied 50% dropout on its fully-connected layers [10]). Dropout forces the network to not rely too heavily on any one neuron, improving robustness. Notably, on CIFAR-10, applying strong regularization like dropout has been shown to yield better results – many top-performing models used dropout to reach minimum error rates [13]. By adding a dropout layer (with rate ~0.3–0.5), we can expect improved generalization and lower test error, especially in combination with the other measures above.

Sources: The recommendations and justifications are drawn from the VGG paper by Simonyan & Zisserman [4] [10], the original ResNet paper by He et al. [1] [8], and the MobileNet paper by Howard et al. [5] [2], as well as standard best practices in deep learning. Each suggested change is feasible within Colab's constraints and should help boost CIFAR-10 test accuracy while keeping the model under the 400k parameter limit.

---

[1] [6] [7] [8] [11] [12] [13] [1512.03385] Deep Residual Learning for Image Recognition
https://ar5iv.org/pdf/1512.03385

[2] [3] [5] [1704.04861] MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications
https://ar5iv.labs.arxiv.org/html/1704.04861

[4] [1409.1556] Very Deep Convolutional Networks for Large-Scale Image Recognition
https://arxiv.org/abs/1409.1556

[9] [10] [1409.1556] Very Deep Convolutional Networks for Large-Scale Image Recognition
https://ar5iv.org/pdf/1409.1556