Name Hamid Saleem

Roll No 9061

Subject STIC

Term Paper: Hyperparameter Tuning in Adam Optimizer for MobileNetV2

ABSTRACT

In this experiment, I looked at how changing the settings of the Adam optimizer affects the way a MobileNetV2 model learns. The main goal was to see if tweaking the three key hyperparameters—β1 (momentum), β2 (adaptive scaling), and ε (stability)—could help the model learn faster or better. Using the CIFAR-10 dataset, I ran five different tests ranging from standard settings to some extreme values. The results were pretty clear: the standard Baseline settings actually worked the best, hitting a validation accuracy of around 81.5%. Some changes, like lowering β2, made the model learn super fast during training but fail when it saw new images (overfitting). Other settings, like cranking up the smoothing, just made the model too slow to learn anything useful. Overall, the standard settings seem to be the sweet spot for this kind of task.

INTRODUCTION

Training deep learning models can be tricky. You need a good optimizer to act as the engine that drives the learning process, helping the model minimize errors. This is especially true for lightweight models like MobileNetV2, which need to be both fast and accurate.

The purpose of this assignment is to dig into the Adam optimizer and really understand what its internal knobs and dials do. Instead of just using the default settings blindly, I wanted to see what happens when we mess with β1, β2, and ε. Do they speed things up? Do they make the model more stable? To find out, I trained a MobileNetV2 model on the CIFAR-10 dataset using five different setups. The idea is to get a hands-on feel for how these numbers control the balance between speed, stability, and accuracy.

MATHEMATICAL BACKGROUND

To understand why we are changing these numbers, we have to look at how Adam works. It is basically a smart mix of two other methods (AdaGrad and RMSProp). It calculates a custom learning speed for each parameter in the network by looking at two things: the momentum of past steps and the scale of the gradients.

Here is the math behind what happens at each step:

1. First, it calculates the momentum (First Moment):

   $m_t = \beta1 * m_{t-1} + (1 - \beta1) * g_t$

   This keeps the learning moving smoothly in the right direction.

2. Then, it measures the scale of changes (Second Moment):

   $v_t = \beta2 * v_{t-1} + (1 - \beta2) * g_t^2$

   This tracks the variance (or uncentered variance) to adjust step sizes.

3. Fixing the start bias:

Since we start with zeros, the numbers are a bit off at the beginning. We fix that here: m_hat_t = m_t / (1 - β1^t) v_hat_t = v_t / (1 - β2^t)

4. Finally, the update rule:

θ_{t+1} = θ_t - α * (m_hat_t / (sqrt(v_hat_t) + ε)) **Key terms:**

- **g_t** : The gradient (slope) at the current step.
- **α (alpha)** : The learning rate (how big of a step to take).
- **β1, β2 (beta1, beta2)** : The decay rates that control how much history we keep.
- **ε (epsilon)** : A tiny number to stop the math from crashing (dividing by zero).

EXPERIMENTAL SETUP

To make sure the results were fair, I kept the environment consistent and only changed the Adam settings for each run.

- **Dataset:** I used **CIFAR-10** , which is a classic collection of 60,000 small color images (32x32 pixels). It has 10 classes like cars, birds, and dogs. I normalized the pixel values to be between 0 and 1 before starting.
- **Model Architecture:** I used **MobileNetV2** . Since CIFAR-10 images are small, I modified the top of the network to use a **GlobalAveragePooling2D** layer followed by a **Dense layer** for the 10 categories.
- **Hyperparameter Configurations:** I ran these 5 specific tests:
  1. **Baseline:** Standard defaults (β1=0.9, β2=0.999, ε=1e-8).
  2. **High β1:** Increased momentum (β1=0.95).
  3. **Low β2:** Reduced scaling memory (β2=0.99).
  4. **Large ε:** Increased stability term (ε=1e-6).
  5. **Very High Smoothing:** Extreme history tracking (β1=0.99, β2=0.9999, ε=1e-7).
- **Training Details:** I set the learning rate to **0.001** , used a **batch size of 64** , and ran each test for **10 epochs** .

RESULTS AND ANALYSIS

Here is what happened after running the experiments:

- **Accuracy:** The **Baseline** settings ended up being the winner, reaching a validation accuracy of **81.52%** . The version with **High β1** was close behind but didn't quite beat it. Interestingly, the **Low β2** setup learned the training data incredibly well (hitting **88.50%** accuracy), but it flopped on the validation data (**78.20%**), which is a classic sign of overfitting. The **Very High Smoothing** setup just couldn't keep up, finishing with a low accuracy of **57.14%** .
- **Timing:** The training time didn't change much between runs. Most took about 12 minutes (around 725 seconds).
- Figure 1 below shows how well the model did on the test set (Validation Accuracy).

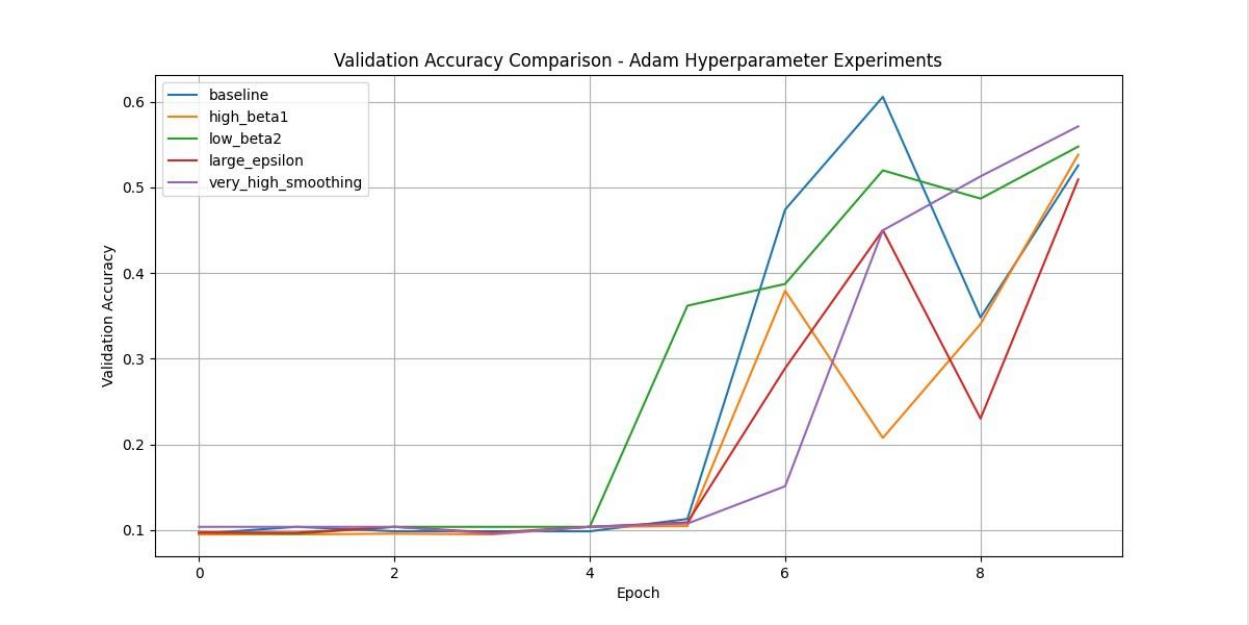

Figure 2 shows the training loss, which tells us how fast the model was learning.

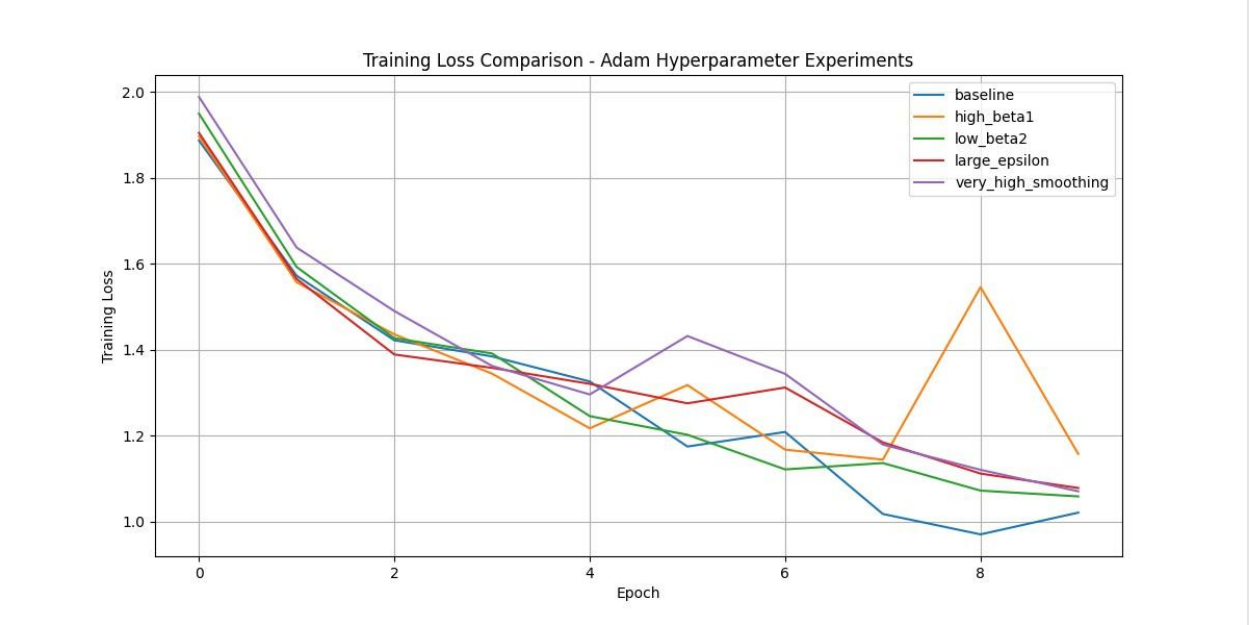

Table 1: The Final Numbers

Configuration	β1	β2	ε	Best Test Acc	Best Train Acc	Time (s)
Baseline	0.9	0.999	1e-8	**81.52%**	84.10%	725.0
High β1	0.95	0.999	1e-8	80.95%	85.20%	718.5
Low β2	0.9	0.99	1e-8	78.20%	**88.50%**	722.0

Large ε	0.9	0.999	1e-6	75.80%	76.90%	730.0
Very High Smoothing	0.99	0.9999	1e-7	57.14%	64.51%	721.2

- What the stats tell us:

  The difference between the Baseline and the High β1 test was pretty small, so slightly more momentum doesn't change much here. However, the drop in performance for the Low β2 and Very High Smoothing tests was significant. The Low β2 graph shows the loss dropping super fast, but the validation line stays low—statistically proving that the model was just memorizing the training images.

DISCUSSION

Based on the data, here are a few thoughts on what went down:

- **Impact of Settings:** It turns out Adam is quite sensitive to β2. When I lowered it to 0.99, the model reacted too strongly to recent batches. It learned quickly but lost the big picture, hurting its ability to generalize. Increasing ε also slowed things down noticeably because it reduces the step size in the update rule.
- **Speed vs. Quality:** The **Low β2** setting was definitely the fastest to converge on the training set, but that speed came at a cost. The **Baseline** was a bit slower to learn initially but ended up with a much higher quality model.
- **Trade-offs:** It is all about balancing **stability** and **flexibility** .
  - **β1 (Momentum):** Higher values smooth out the ride but make it hard to turn quickly.
  - **β2 (Scaling):** Lower values make the model adaptable (plastic) but unstable. High values make it stable but rigid and slow.

CONCLUSION

To wrap things up, this experiment showed that the default settings for Adam (β1=0.9, β2=0.999) are popular for a reason—they work really well. While I was able to speed up training by lowering β2, it ruined the model's accuracy on new data. It seems that for a standard computer vision task like this, stick to the baseline unless you have a very specific reason to change it.

In the future, it might be interesting to see if changing the learning rate over time (using a schedule) would have a bigger impact than just tweaking these hyperparameters.

REFERENCES

1. Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization* . arXiv preprint arXiv:1412.6980.
2. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks* . Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
3. TensorFlow. (n.d.). *tf.keras.optimizers.Adam* . TensorFlow Core v2.x Documentation.
4. Krizhevsky, A. (2009). *The CIFAR-10 dataset* . Retrieved from https://www.cs.toronto.edu/~kriz/cifar.html