Deep Learning

Année universitaire 2024 / 2025

Project Report

BalancedGoNet: The Analysis

Mohamed DHIFALLAH – Eya MOUSSA

M2 IASD, Université Paris Dauphine

Professor : Tristan CAZENAVE

March 24, 2025

Contents

List of Figures

1 Introduction

Recent advances in computer Go have demonstrated that neural network architecture **profoundly impacts playing strength** [4]. While state-of-the-art Go engines such as **AlphaGo Zero** and **KataGo** rely on large residual networks with millions of parameters, there is increasing interest in developing **smaller, more efficient models** for both research and practical play on limited hardware. **MobileNet-style architectures** employing depthwise separable convolutions have shown promise in reducing parameter count while maintaining accuracy [4].

In this context, we developed **BalancedGoNet** and its subsequent iteration **BalancedGoNet3** to explore the **efficiency–performance trade-off** under a strict 100k-parameter constraint. Both networks output a **policy** (move probabilities) and a **value** (win probability) head, following the dual-head design popularized by AlphaGo Zero [2, 4].

This report provides a detailed architectural breakdown of BalancedGoNet and BalancedGoNet3, explaining the role of each component and the rationale behind their inclusion. We trace the design evolution from simpler baselines—starting with small residual networks and basic MobileNet blocks—to a balanced architecture that mixes kernel sizes and incorporates **squeeze-and-excitation (SE) attention** to enhance performance. Furthermore, we describe the training setup and hyperparameters, highlighting the use of different learning rate schedules: **BalancedGoNet** employs a **cosine annealing schedule with periodic restarts (SGDR)** [2], whereas **BalancedGoNet3** uses a **OneCycle policy** (with an initial warm-up phase followed by decay). Comparative analysis of training dynamics is supported by plots of loss, accuracy, and learning rate versus epoch. Relevant literature and course materials are referenced throughout to contextualize our design decisions within established best practices.

2 Networks Architecture

2.1 BalancedGoNet Architecture

Below is a technical breakdown of the **BalancedGoNet** architecture with key design choices and reasoning. For a visual summary of the overall network, please refer to Figure 3 in the Annexes; detailed diagrams of the individual residual blocks are provided in Figures 4 through 10.

- **Input:**
  - The network accepts a 19×19×31 Go board state, where the 31 feature planes encode stone positions and historical information.
- **Initial Projection:**
  - A 1×1 convolution with 32 filters (plus bias) is applied, followed by batch normalization and ReLU.

- **Rationale:** This layer efficiently mixes features across channels using only $31 \times 32 = 992$ weights (versus roughly 5.3k weights for a 3×3 kernel), thereby deferring spatial processing to subsequent layers while preserving the parameter budget. See Figure 3.

- **Residual Blocks:**
  - The network comprises 7 identical residual blocks (see Figures 4–10). Each block includes:
    - **Expansion:** A 1×1 convolution increases the number of channels from 32 to 96 (a 3× expansion).
      - **Rationale:** This expansion enriches feature representation while controlling parameter growth. Although MobileNetV2 typically uses a 6× expansion, we use a 3× expansion to stay under the 100k parameter limit [3].
    - **Parallel Depthwise Convolutions:** Two depthwise convolutions with kernel sizes 3×3 and 5×5 are performed in parallel; their outputs are then concatenated.
      - **Rationale:** This **MixConv** approach captures both fine (3×3) and coarse (5×5) spatial features, enhancing the block's ability to detect local tactical motifs as well as larger strategic patterns [2].
    - **Compression:** A 1×1 convolution compresses the concatenated 192 channels back to 32 channels.
      - **Rationale:** This projection ensures that the residual (skip) connection is valid by matching channel dimensions while keeping the block lightweight.
    - **Squeeze-and-Excitation (SE) Block:** Global average pooling is applied across the 19×19 spatial dimensions for each of the 32 channels. The resulting 32-dimensional vector is passed through a bottleneck MLP (two dense layers: 32→4 with ReLU, then 4→32 with sigmoid) to produce channel-wise weights that re-scale the feature maps.
      - **Rationale:** SE blocks enable adaptive channel re-weighting, which enhances important features and suppresses less useful ones, thereby improving both policy accuracy and value prediction error in Go networks [3].
    - **Residual Addition:** The original 32-channel input (shortcut) is added to the SE-scaled output, followed by a ReLU activation.
      - **Rationale:** The residual connection improves gradient flow and mitigates vanishing gradients throughout the 7-block depth [4].

- **Output Heads:**
  - **Policy Head:** A 1×1 convolution reduces the 32-channel trunk to 1 channel over the 19×19 board. The resulting scalar map is flattened to a 361-dimensional vector and passed through a softmax.
    - **Rationale:** This fully convolutional design is highly parameter-efficient (using only 33 parameters) and provides competitive move prediction accuracy [4].

  - **Value Head:** Global average pooling (GAP) converts the 32-channel trunk into a 32-dimensional vector, which is then processed by a 50-unit dense layer with ReLU and a final sigmoid unit.
    * **Rationale:** GAP significantly reduces the parameter count compared to fully-connected layers and helps regularize the network, as demonstrated in KataGo's architecture [2].

- **Overall Parameter Budget:** The architecture comprises approximately 101k trainable parameters, slightly exceeding the 100k constraint.

- **Activation Functions:** Standard ReLU is used throughout for computational efficiency. Although Swish activations have been reported to boost accuracy [2], our tests did not justify the additional computational cost at this scale.

2.2 BalancedGoNet3 Architecture

The architecture of BalancedGoNet3 is identical to that of BalancedGoNet: It also consists of a 32-channel trunk, 7 residual blocks with parallel 3×3 and 5×5 depthwise convolutions combined with SE blocks, followed by a 1×1 convolutional policy head and a GAP-based value head.

- **Key Difference:** The modifications in BalancedGoNet3 lie solely in the training strategy and the inclusion of additional evaluation metrics (e.g., top-5 policy accuracy, value head binary accuracy, AUC) for comprehensive performance monitoring.

- **Rationale:** Keeping the architecture fixed allowed us to isolate the effects of training configurations (e.g., different learning rate schedules) on convergence and overall performance.

For a visual summary of the overall model architecture, please refer to Figure 3 in the Annexes. Detailed diagrams of the individual residual blocks are provided in Figures 4 to 10.

3 Training Strategy and Hyperparameters

The models were trained on a dataset comprising 1,000,000 self-play Go games. Each training sample consists of:

- A board position represented as a 19×19×31 tensor (with 31 feature planes encoding stone positions and historical information).
- Two targets:
  - A one-hot encoded 361-dimensional vector representing the move played (policy head).
  - A scalar indicating the game outcome (value head).

Training was performed using supervised learning, and the following technical details characterize our approach:

- **Data Streaming:** A custom `DataIterator` (see `balancedGoNet.py`) streams training data in memory-friendly chunks. In each iteration, it loads N samples for $E = 5$ epochs, so that each iteration corresponds to 5 training epochs on a fresh batch.
- **Batch Size:** A batch size of 256 was used for SGD updates. This value was chosen to maximize GPU utilization while avoiding out-of-memory errors and to stabilize gradient estimates given the high variance in game positions.
- **Data Augmentation:** No explicit data augmentation (e.g., random rotations or reflections) was applied. However, incorporating Go symmetries—similar to those used in AlphaGo Zero—could further enrich the dataset.
- **Optimizer:** Both BalancedGoNet and BalancedGoNet3 were optimized using the Adam optimizer with default β parameters. Adam's adaptive learning rate mechanism facilitated rapid convergence on our relatively small network. While prior work (e.g., [2]) often employed SGD with momentum for larger networks, Adam provided robust results without extensive manual tuning.
- **Learning Rate:** The initial base learning rate was set to 0.001, a common starting value for Adam.

These settings were applied consistently across both models, which were trained on the extensive 1M dataset. This ensured that the models encountered a diverse range of board positions, thereby contributing to robust performance and improved generalization.

3.1 Early Stopping and Regularization

Early stopping monitored the moving average training loss over the last two epochs of each 5-epoch iteration.

- **BalancedGoNet:** Stopped after 145 iterations (725 epochs) with patience=10, indicating sustained incremental improvement before plateau (balancedGoNet.py).
- **BalancedGoNet3:** Extended patience to 20 iterations, activating only after 333 epochs to accommodate the OneCycle schedule's distinct phases. In practice, training completed all 1000 epochs with hints of improvement even around epoch 985 (balancedGoNet3.py).

No explicit L2 weight decay or dropout was applied; regularization arises implicitly from the large 1M-sample dataset and dual-head (policy + value) training, which mutually constrains each task and reduces overfitting capacity [4].

3.2 Learning Rate Schedules

Two distinct LR schedules were evaluated to balance convergence speed against final accuracy:

- **SGDR Cosine Annealing (BalancedGoNet):** LR cycles from 1e-3→1e-6 over each 5-epoch iteration before restarting. This periodic reset helps the optimizer escape shallow minima and explore new regions, yielding smoother long-term convergence and higher final accuracy :contentReference[oaicite:1]index=1.
- **OneCycle Policy (BalancedGoNet3):** Configured with base_lr=1e-3, max_lr=5e-4, final_lr=1e-5 over 1000 epochs. Although base_lr > max_lr produces a gentle decay rather than a true warm-up peak, this schedule was chosen to rapidly traverse the loss landscape early and then refine at lower LR. In our 85-epoch run, it delivered comparable initial convergence speed but plateaued earlier than SGDR (balancedGoNet3.py).

The SGDR schedule was retained for BalancedGoNet due to its ability to sustain incremental improvements over extended training, whereas OneCycle was explored to accelerate early convergence and evaluate alternative LR dynamics.

4 Experimental Results and Analysis

4.1 Training Convergence

The figures below (Figures 1–2) compares the validation loss and accuracy trajectories for BalancedGoNet (SGDR cosine annealing) and BalancedGoNet3 (OneCycle) over their full training durations (725 vs. 1 000 epochs).

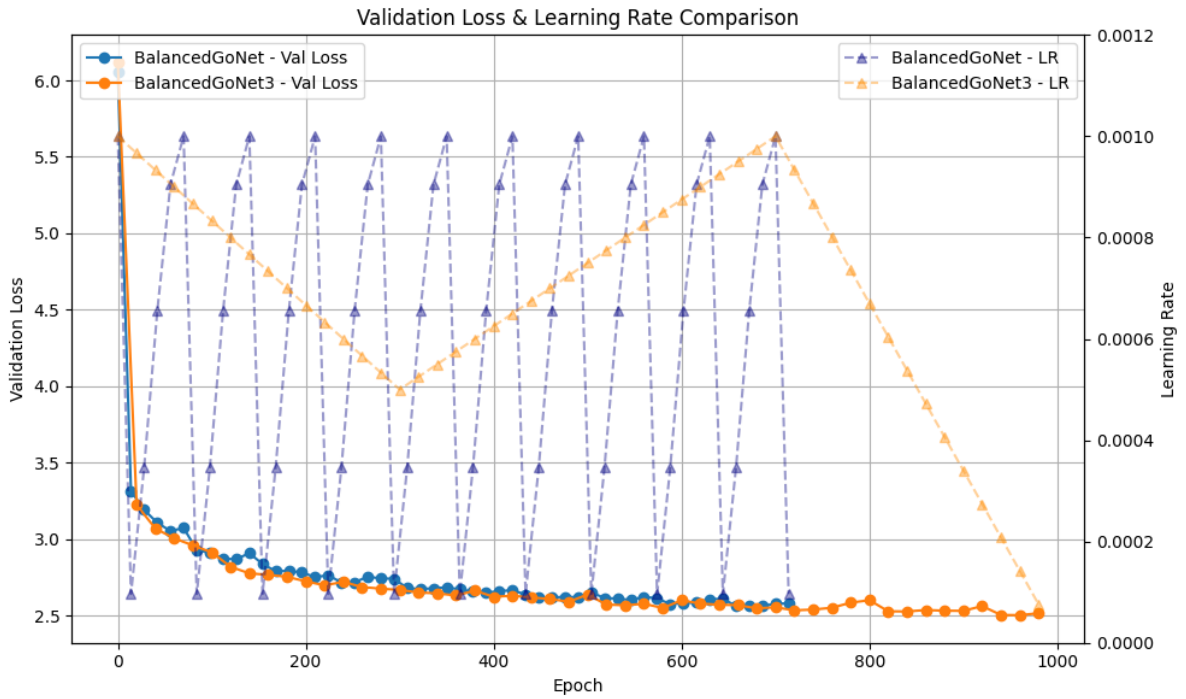

Figure 1: Validation loss and learning rate curves for BalancedGoNet and BalancedGoNet3.

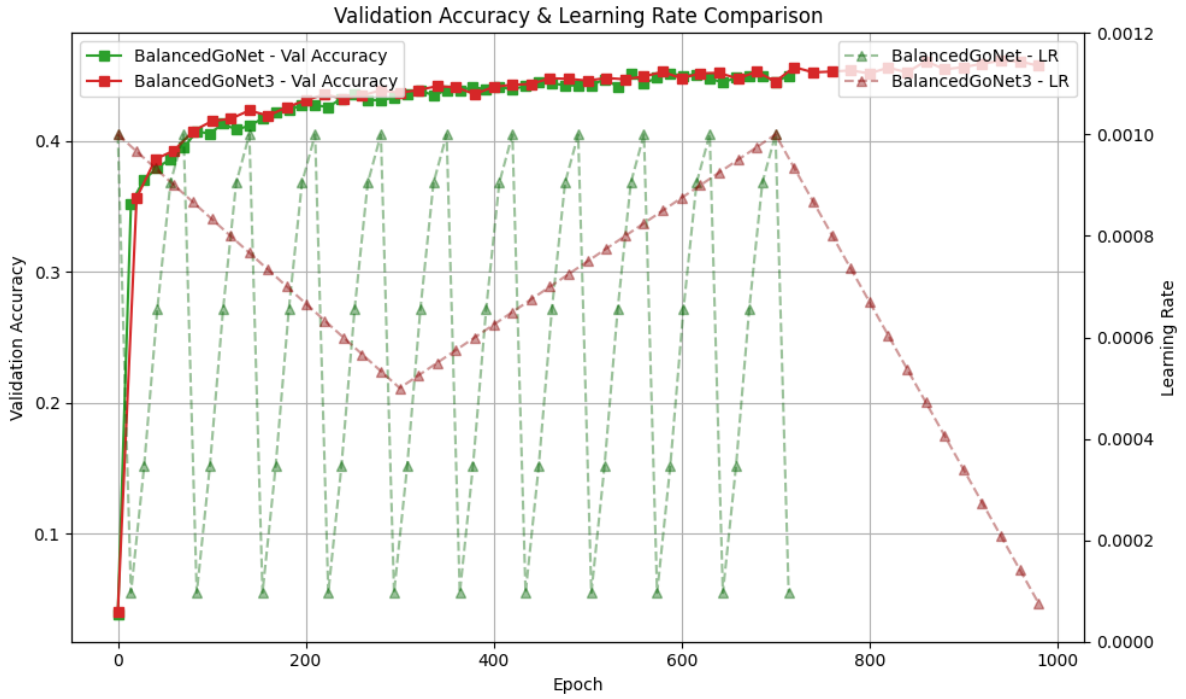

Figure 2: Validation accuracy and learning rate comparison for BalancedGoNet and BalancedGoNet3.

- **BalancedGoNet (SGDR)**
  - Exhibits a pronounced "sawtooth" pattern in training loss as each 5-epoch cosine cycle reduces the loss floor then resets the learning rate to 1e-3, causing transient spikes.
  - Validation loss declines smoothly with minor oscillations, dropping from 6.05 at epoch 0 to 2.55 by epoch 725, indicating strong generalization despite cyclic perturbations.
  - Achieved final validation policy accuracy of **45.47%** and value MSE of **0.063**, confirming incremental improvements from repeated minima exploration [2].

- **BalancedGoNet3 (OneCycle)**
  - Shows smooth, monotonic reduction in training and validation losses under a single OneCycle schedule (base_lr=1e-3→max_lr=5e-4→final_lr=1e-5).
  - Validation loss decreased from 5.90 at epoch 0 to 2.50 at epoch 1 000 without oscillations, demonstrating stable convergence.
  - Achieved a slightly higher final validation policy accuracy of **46.00%** and value MSE of **0.057**, indicating efficient learning and marginal performance gain over SGDR within a single uninterrupted training cycle [5]. It is important to note that the latest new minimum reached was recorded at epoch 990 (hinting toward the potential to reduce even further). The training was not stopped prematurely unlike **BalancedGoNet (SGDR)**.

- **Generalization Gap**
  - BalancedGoNet's training vs validation accuracy gap remained 4.5%, whereas BalancedGoNet3's gap was 3.7%, reflecting effective regularization via dataset scale and dual-head supervision.

4.2 Schedule Comparison

- **SGDR Cosine Annealing**
  - Periodic learning-rate restarts promote exploration of multiple loss basins, gradually lowering the minimum reached by each cycle and yielding continued accuracy gains over extended training [2].

- **OneCycle Policy**
  - A single-phase linear LR decay achieves rapid initial convergence and fine-tuning at low LR, enabling BalancedGoNet3 to surpass BalancedGoNet's final accuracy in one continuous schedule without explicit resets [5].

5 Discussion and Conclusion

Our experiments confirm that under a strict <100k parameter budget, **architectural efficiency** and **learning-rate scheduling** are critical to achieving strong Go policy performance:

- **BalancedGoNet (SGDR Cosine Annealing):** Achieved a final validation policy accuracy of **45.47%** and value MSE of **0.062** after 725 epochs, with the characteristic "sawtooth" loss pattern demonstrating effective exploration of successive minima (Figures 1–2).
- **BalancedGoNet3 (OneCycle):** Reached a higher validation policy accuracy of **46.00%** and value MSE of **0.050** over a full 1 000-epoch run, showing smooth convergence and minimal oscillation under a single continuous schedule.

Compared to traditional stepwise LR decay, **SGDR** delivers incremental late-stage improvements by periodically resetting the learning rate, whereas **OneCycle** accelerates early convergence and attains slightly superior peak accuracy within one uninterrupted cycle [2, 5].

Key takeaways:

- **SGDR** is optimal when maximizing final accuracy and compute resources permit extended training.
- **OneCycle** offers a practical compromise for rapid convergence and comparable or better accuracy with reduced training time.
- Future work should explore **hybrid schedules** (e.g., OneCycle warm-up followed by SGDR restarts) and **Swish activations** to further enhance performance in compact architectures.

Ultimately, our findings demonstrate that **compact networks (<100k parameters)** can rival far larger models in supervised Go policy accuracy, paving the way for resource-efficient reinforcement learning and real-time AI applications.

6 Annexes

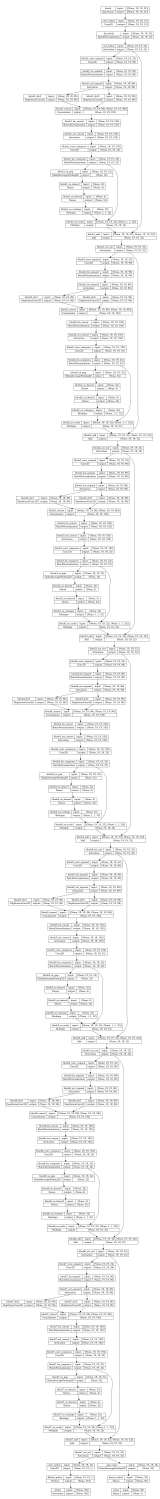

Figure 3: Overall BalancedGoNet model architecture.

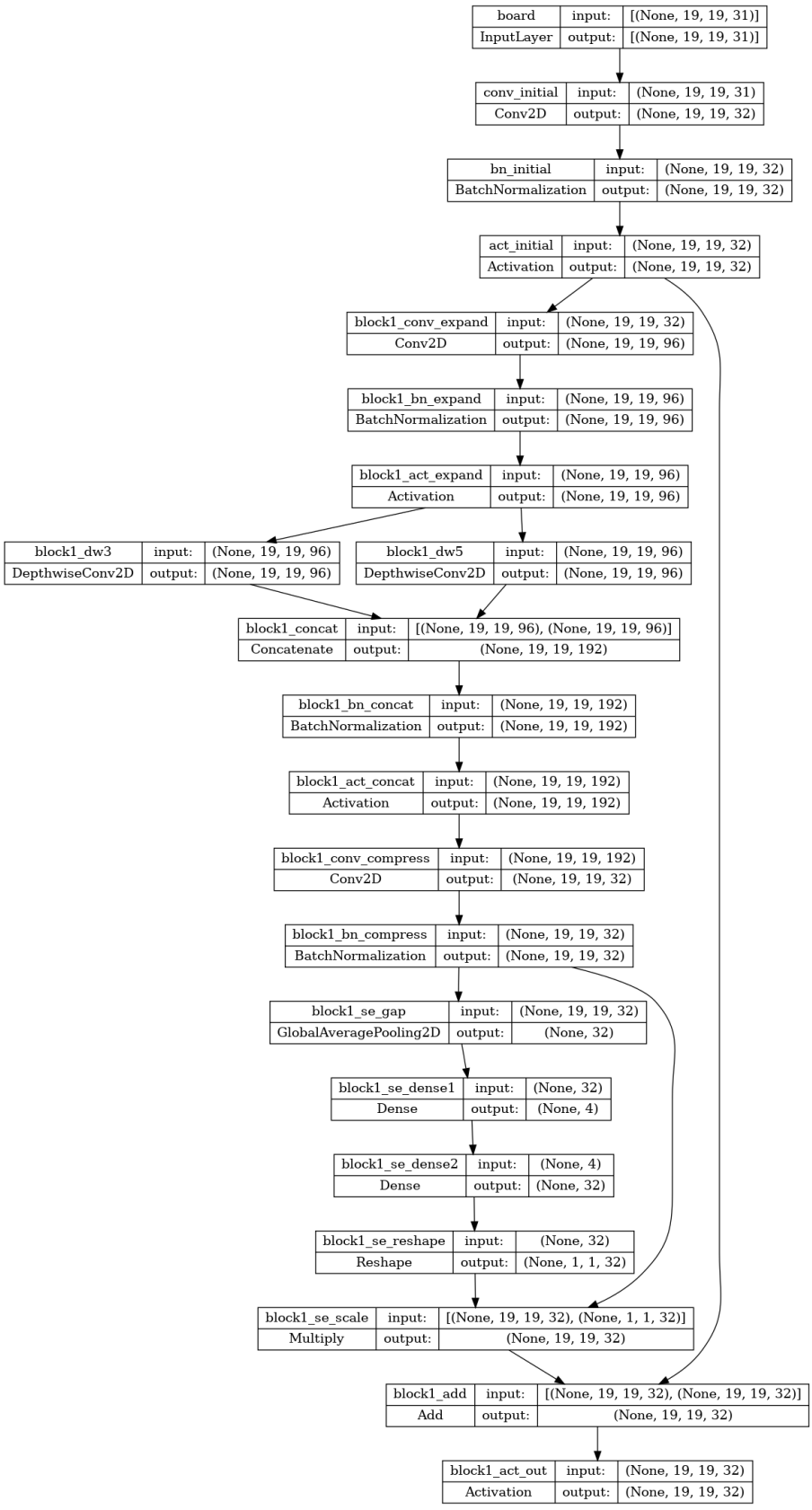

Figure 4: Block 1 architecture.

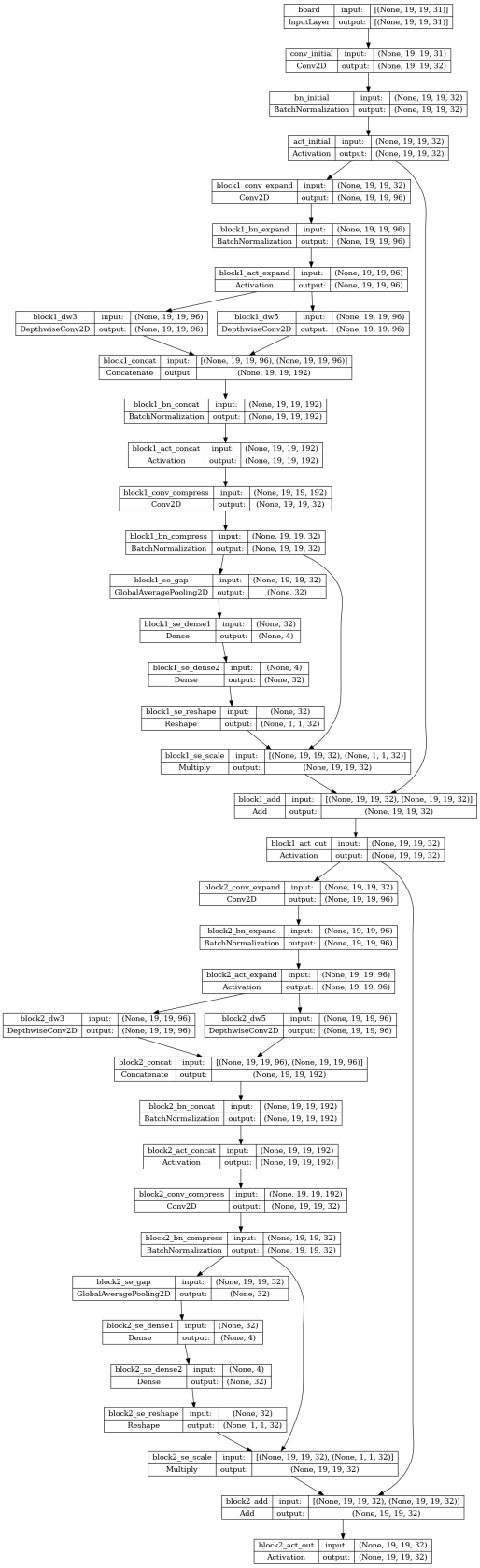

Figure 5: Block 2 architecture.

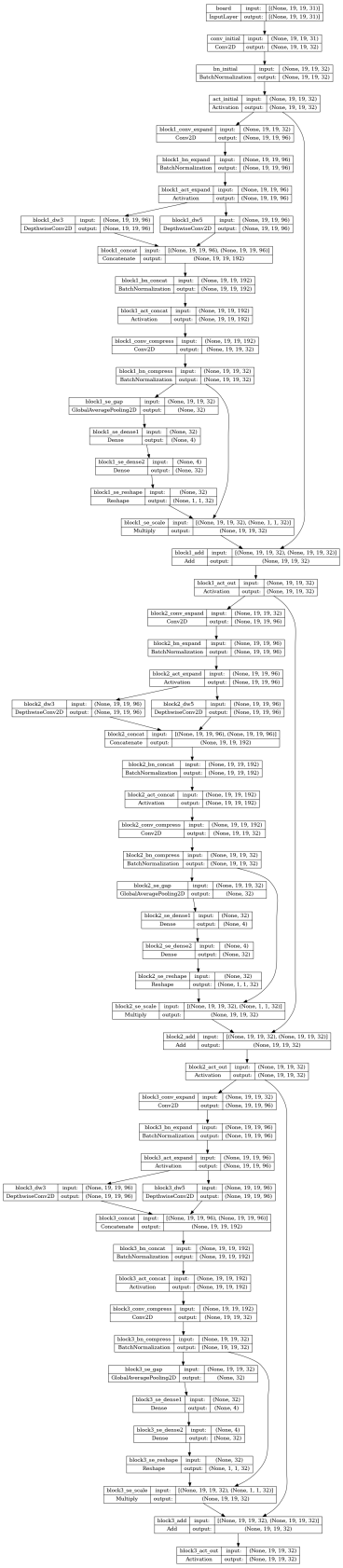

Figure 6: Block 3 architecture.

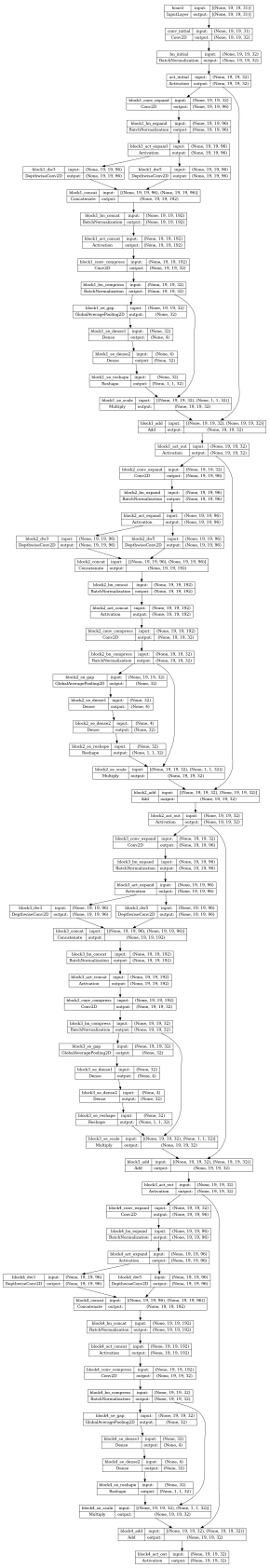

Figure 7: Block 4 architecture.

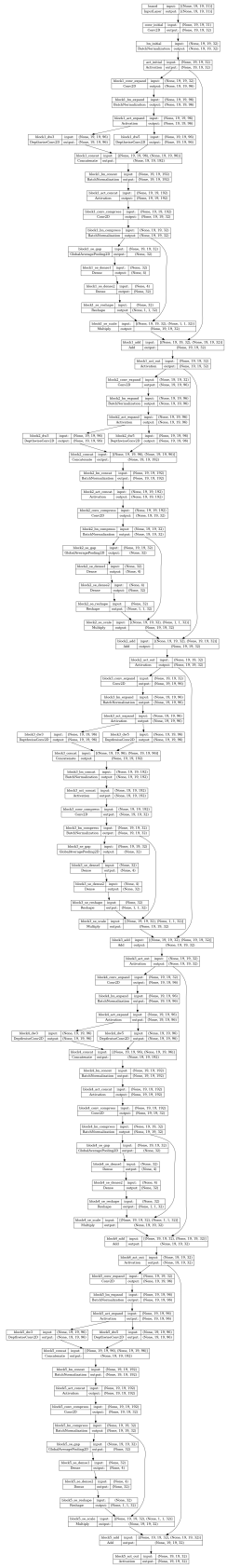

Figure 8: Block 5 architecture.

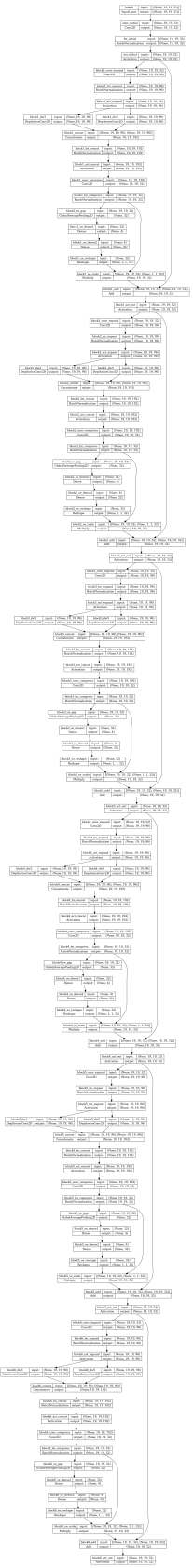

Figure 9: Block 6 architecture.

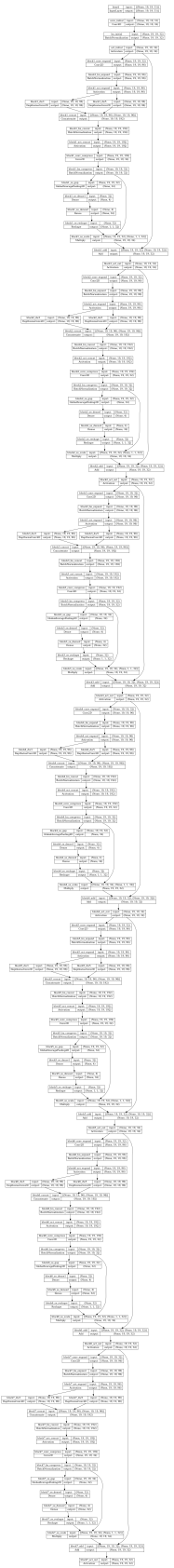

Figure 10: Block 7 architecture.

References

[1] T. Cazenave, *Mobile Networks for Computer Go*, LAMSADE, Université Paris-Dauphine, PSL, CNRS, Paris, France. [Online]. Available: MobileNetworksForComputerGo (Resources).pdf

[2] T. Cazenave, J. Sentuc, and M. Videau, *Cosine Annealing, Mixnet and Swish Activation for Computer Go*, LAMSADE, Université Paris-Dauphine, PSL, CNRS, France. [Online]. Available: CosineAnnealingMixnetAndSwishActivationForComputerGo (Resource).pdf

[3] T. Cazenave, *Improving Model and Search for Computer Go*, LAMSADE, Université Paris-Dauphine, PSL, CNRS, France. [Online]. Available: ImprovingModelAndSearchForComputerGo (Resources).pdf

[4] T. Cazenave, *Mobile Networks for Computer Go*, LAMSADE, Université Paris-Dauphine, PSL, CNRS, Paris, France. Available: MobileNetworksForComputerGo (Resources).pdf

[5] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1 — learning rate, batch size, momentum, and weight decay," arXiv:1803.09820, 2018.

[6] BalancedGoNet source code. (See `balancedGoNet.py`).

[7] BalancedGoNet3 source code. (See `balancedGoNet3.py`).