NanoGo: Exploring Lightweight Architectures for Efficient Go Playing Networks

Mouhamed SY

PSL Research University, Master IASD
Teacher: Tristan Cazenave

March 23, 2025

Abstract

This work investigates the design of lightweight deep learning architectures for the game of Go under strict computational constraints. Using high-quality self-play data from KataGo, we explore and benchmark several compact neural networks each constrained to approximately 100,000 parameters for the dual tasks of move prediction (policy) and game outcome estimation (value). Our study evaluates classical convolutional models such as RESNet and MobileNet, as well as more recent architectures including Inception with attention, MNASNet, normalization-free Transformers, and hybrid CNN-attention models such as Linformer and EfficientFormer. Each model is customized for 19×19 Go boards, with consistent training pipelines, dual output heads, and optimized learning rate schedules. Through extensive experimentation, we highlight trade-offs in model expressiveness, training stability, and parameter efficiency. Our results provide practical insights into deploying compact and interpretable models for strategic board games.

Index Terms— Deep Learning, Attention Mechanisms, Computer Go , Model Efficiency.

1 Introduction

The game of Go has long been a benchmark challenge for artificial intelligence due to its vast search space and deep strategic complexity. Unlike games such as chess, where brute-force search has been effective, Go requires a more sophisticated balance of intuition and precise calculation. The introduction of deep reinforcement learning (DRL) combined with Monte Carlo Tree Search (MCTS) has revolutionized AI performance in Go, leading to the development of superhuman-level agents such as AlphaGo (Silver et al., 2016), AlphaZero (Silver et al., 2018), and KataGo (Wu et al., 2019). These models leverage deep convolutional neural networks to evaluate board states and guide MCTS, significantly improving the efficiency of decision-making. However, their success comes at the cost of high computational demands, making them impractical for deployment on resource-constrained hardware.

To address this challenge, a Deep Learning project was organized as part of the IASD Master program at PLS Rsearch University, with the objective of designing efficient neural networks to play Go under strict computational constraints. The competition required students to train neural networks with fewer than 100,000 parameters, ensuring fairness in training resources while encouraging research into lightweight AI architectures. Each team, composed of up to two students, had access to a dataset of 1,000,000 self-played games generated by KataGo. The

input representation consists of 31 feature planes (19×19) encoding board states, ladders, and move history. The models are trained to predict:

- **A policy head**: a probability distribution over the 361 board positions, indicating the move played.
- **A value head**: a scalar prediction of the game outcome, close to 1.0 if White wins and 0.0 if Black wins.

To facilitate training and evaluation, students were provided with **Golois**, a python library programmed in C++ that enables efficient processing of game states and neural network training. Golois is optimized for handling large datasets and integrates seamlessly with PyTorch and TensorFlow, allowing students to experiment with different architectures efficiently. Additionally, an MCTS-based evaluation system was implemented to enable the trained networks to compete against each other in a round-robin tournament, assessing their relative performance in actual gameplay.

As part of this initiative, NanoGo aims to systematically evaluate lightweight neural architectures for playing Go under the imposed 100,000-parameter constraint. Traditional Go-playing networks, such as those used in AlphaZero and KataGo, typically follow a residual network (ResNet) structure. While ResNets have demonstrated strong performance, alternative architectures optimized for efficiency, such as MobileNet (Howard et al., 2017), models based on MNASNet (Tan et al., 2019), and Inception-based architectures (Szegedy et al., 2015), have been widely adopted in mobile AI applications.

Beyond convolutional networks, we also explore attention-based mechanisms to determine whether they can improve performance under strict resource constraints. In particular, we evaluate Linformer (Wang et al., 2020), a lightweight Transformer variant that reduces the quadratic complexity of self-attention to $O(n)$, making it more efficient for structured input representations like Go board states. By comparing Linformer to convolutional approaches, we aim to understand whether attention-based architectures provide an advantage in capturing long-range dependencies within the game. Training is conducted on 1,000,000 self-played games from KataGo, using 31 input feature planes encoding board states, ladders, and historical moves. The models are evaluated based on policy accuracy, value prediction error (MSE), inference speed, and computational efficiency, ensuring an optimal trade-off between performance and resource constraints.

By addressing the challenges of efficiency in deep learning for strategic games, NanoGo contributes to the development of compact AI models that maintain strong decision-making capabilities while operating under stringent computational limitations. The findings may extend beyond Go, offering insights applicable to other board games and low-resource AI applications, such as mobile AI, embedded systems, and on-device learning.

2 Background

The game of Go has historically been one of the most challenging problems in artificial intelligence due to its immense search space and the necessity for deep strategic reasoning. Unlike chess, where brute-force search combined with heuristics has proven highly effective, Go requires a more intricate balance between intuition, local tactical calculations, and long-term strategic planning. With an estimated game-tree complexity of 10^{170}, Go vastly surpasses other classical board games in terms of combinatorial possibilities. Early AI systems for Go were primarily based on handcrafted rules and Monte Carlo methods, but these approaches were limited in their ability to generalize across different positions. The introduction of Monte Carlo Tree Search (MCTS) (Coulom, 2006) marked a pivotal moment in Go AI development, as it enabled efficient exploration of game states through statistical simulations

and dynamic move selection. The integration of Deep Reinforcement Learning (DRL) with MCTS represented another milestone, leading to the development of AlphaGo (Silver et al., 2016), AlphaZero (Silver et al., 2018), and KataGo (Wu et al., 2019). These models employed deep convolutional neural networks (CNNs) to evaluate board positions and guide MCTS, dramatically improving gameplay strength. While AlphaGo used a combination of supervised learning and reinforcement learning, AlphaZero introduced a fully self-play training paradigm, achieving even higher performance with a more streamlined architecture. KataGo further optimized the model by incorporating adaptive training techniques and faster search heuristics, making it one of the most advanced open-source Go AIs available today.

Traditional Go AI architectures have largely been based on Residual Networks (ResNets) (He et al., 2016), which provide a deep hierarchical structure capable of extracting complex patterns from board states. However, these models often require millions of parameters, making them computationally expensive and difficult to deploy on low-resource hardware. To address these limitations, research has focused on developing lightweight architectures that maintain strong predictive capabilities while reducing computational cost. Studies have explored **MobileNet** (Howard et al., 2017) and **MNASNet** (Tan et al., 2019), which leverage depthwise separable convolutions to improve efficiency. Additionally, **MixNet-based architectures**, which combine multiple kernel sizes within convolutional layers, have been shown to enhance accuracy with minimal parameter overhead (Cazenave et al., 2021).

To further enhance the representational power of lightweight models, attention mechanisms such as **squeeze-and-excitation (SE) blocks** (Hu et al., 2018) have been introduced. SE blocks adaptively recalibrate channel-wise feature responses by explicitly modeling interdependencies between channels. This mechanism allows the network to focus on more informative features while suppressing less useful ones, thereby improving performance without significantly increasing the parameter count. SE modules have been successfully integrated into various architectures—including ResNet, MobileNet, and MixNet variants—demonstrating consistent gains in accuracy for a modest computational overhead.

Beyond convolutional approaches, transformer-based architectures have emerged as an alternative paradigm for Go AI. Unlike CNNs, which process information locally through convolutional filters, Vision Transformers (ViTs) rely on self-attention mechanisms to capture long-range dependencies between board positions. Recent studies have investigated the potential of **EfficientFormer** (Sagri et al., 2023) and **Linformer** (Wang et al., 2020), which optimize the computational complexity of transformers to make them more feasible for structured input representations like Go board states. While transformers have shown promising results in various domains, their effectiveness in Go remains an active area of research compared to CNN-based architectures.

In addition to network architecture, optimization techniques play a crucial role in improving training efficiency and model convergence. Strategies such as **cosine annealing learning rate schedules** (Cazenave et al., 2021) and alternative activation functions like **Swish** (Ramachandran et al., 2017) have demonstrated improvements in training stability and overall model performance. Furthermore, advancements in search algorithms have contributed to stronger AI performance, with the PUCT (Predictor + Upper Confidence Bound for Trees) algorithm remaining the standard for move selection. Recent work on Generalized PUCT (GPUCT) has proposed refinements to search efficiency across different computational budgets (Cazenave, 2021).

Despite these advancements, achieving a balance between computational efficiency and strategic depth remains a challenge in Go AI. While lightweight models such as MobileNet and MNASNet enhance inference speed, they often experience accuracy degradation compared to deeper architectures like ResNets. Transformer-based models offer an alternative by capturing long-range dependencies, yet their feasibility in Go remains under investigation. Future research could explore hybrid approaches that integrate convolutional and attention mechanisms to

optimize feature extraction.
In addition to architecture design, advances in knowledge distillation (Hinton et al., 2015) and progressive model pruning may further improve efficiency by transferring knowledge from larger models to compact networks. Moreover, refining MCTS enhancements, such as adaptive search policies or dynamic exploration-exploitation trade-offs, could provide additional performance gains. As AI research in strategic games evolves, the study of efficient learning paradigms will be essential for deploying high-performance models in computationally constrained environments.

3 Network Architectures and Optimization

In this section, we describe the methodology used to design and evaluate lightweight neural networks for playing Go, while strictly adhering to a parameter budget of 100,000. The primary objective is to explore and compare various neural network architectures and optimization strategies in order to identify the most effective model that balances playing strength and computational efficiency. We present the different architectures tested, outline the training procedures, and detail the techniques employed to improve performance within the given constraints.

3.1 Dataset and Problem Formulation

The data used for training comes from self-played games generated by KataGo (Wu, 2021), currently one of the strongest available computer Go programs. In 2020, KataGo released a large collection of games in SGF format. Each input in this dataset is represented as 31 feature planes of size 19×19, capturing information such as the color to play, ladder patterns, the current board state (on two planes), and two previous board states (each represented using four planes). These features provide both temporal and tactical context to the neural network.
The outputs are twofold: the policy, represented as a 361-dimensional vector (one value per board position), where the correct move is labeled with 1.0 and all others with 0.0; and the value, a scalar close to 1.0 if White wins, and close to 0.0 if Black wins. For validation, a separate dataset was constructed by randomly selecting 10,000 games from the 1,000,000-game training set, and sampling one random board state from each game. These validation states are strictly excluded from the training process to ensure unbiased evaluation.
Compared to datasets used in previous works such as those based on ELF OpenGo (Tian et al., 2019) and Leela Zero (Pascutto, 2017), the KataGo dataset offers significantly higher-quality training data. KataGo consistently plays at a stronger level than both ELF and Leela, and networks trained on this dataset benefit from richer and more accurate examples of expert-level Go play.

3.2 Training

All models were trained using TensorFlow 2.15 on an NVIDIA A100 GPU, offering a good compromise between performance and resource efficiency. To ensure reproducibility and fair comparison between architectures, the training pipeline followed a consistent and modular structure. Each experiment was implemented using Keras' functional API, and conformed to the same training protocol under a strict parameter budget of 100,000. The typical training structure began with the configuration of hyperparameters, such as the number of filters, residual blocks, regularization strength, batch size, and total number of epochs (set to 500). The model architecture included an input convolutional stem (merging 5×5 and 1×1 convolutions), followed by a stack of residual blocks, and two output heads: a policy head producing a 361-dimensional probability distribution via softmax, and a value head predicting the win probability through a sigmoid-activated scalar output. The models were compiled with the Adam optimizer and a dual-loss setup combining categorical cross-entropy for the policy head

and binary cross-entropy for the value head. Evaluation metrics included categorical accuracy and mean squared error (MSE). Dropout and L2 regularization (with a typical strength of 10^{-5}) were applied to prevent overfitting given the small model sizes. Two learning rate scheduling strategies were evaluated to improve convergence and model generalization. The first was a manually defined **piecewise schedule**, where the learning rate was reduced step-by-step at fixed epoch intervals. The learning rate started at 0.005 before epoch 100, dropped to 0.0005 before epoch 150, and continued decreasing progressively down to 5×10^{-8} beyond epoch 400. This simple yet effective scheduling allowed us to tune the learning dynamics at each training stage with full control. The second approach was **cosine annealing** with warm restarts (Loshchilov et al., 2016), designed to provide smoother and more adaptive learning rate decay. The learning rate at epoch t followed the formula:

$$\mathrm{lr}(t) = \frac{\mathrm{lr}_{\mathrm{init}}}{2}\left(\cos\left(\frac{\pi(t \bmod T)}{T}\right) + 1\right)$$

where $\mathrm{lr}_{\mathrm{init}} = 10^{-3}$ and $T = 50$ is the cycle length. This technique enables gradual reductions followed by restarts, helping the model escape sharp local minima and maintain better generalization throughout long training sessions. Training progress was monitored over 500 epochs using TensorBoard, and metrics were also saved manually via dictionaries serialized into `.pkl` files. Early stopping with a patience of 20 epochs was employed to prevent overfitting, and the best model was saved using the `ModelCheckpoint` callback based on validation loss. Every 20 epochs, the models were evaluated on a held-out validation set consisting of 10,000 board positions selected from the original KataGo dataset. This protocol ensured unbiased performance tracking across all training runs and allowed consistent benchmarking of different architectural choices. This robust and reproducible setup enabled systematic exploration of lightweight neural architectures and optimization strategies, with a focus on maximizing playing strength within strict computational and parameter constraints.

3.3 Architectures Explored

In order to identify the most effective neural network architecture under a strict parameter budget of 100,000, we explored a diverse set of lightweight models drawn from recent advances in efficient deep learning.

Our goal was not only to test high-performing architectures from the literature, but also to evaluate their adaptability to the specific task of predicting policy and value outputs in the game of Go. Each architecture was re-implemented and scaled down to meet our parameter constraint, with necessary adjustments to accommodate dual output heads and 19×19 board inputs.

The models explored include: **RESNet**, a compact design using grouped convolutions; **MobileNet**, known for its use of depthwise separable convolutions; **Inception**, leveraging multi-scale convolutional paths; **MNASNet**, a NAS-discovered efficient network; **Normalization-Free Networks (NFN)** (Zhu et al., 2025) , which omit batch normalization entirely and instead rely on scaled weight standardization and activation reparameterization for training stability ; a **Hybrid CNN + Linformer** combining local convolutions with linear attention; and **EfficientNetFormer**, a recent hybrid transformer designed for high accuracy in resource-constrained settings.

Each of these architectures was adapted with custom input stems and go-specific output heads. The following sections provide a brief overview of the design and adaptation choices for each model. A comparative analysis of their performance is presented in Section 4.

3.3.1 RESNet

Residual networks (RESNet) have become a cornerstone of deep learning in strategic board games such as Go. Their introduction into AlphaGo (Silver et al., 2016) enabled the use of deeper architectures by stabilizing training through identity skip connections. Subsequent works, such as (Cazenave, 2020b), confirmed their effectiveness for policy prediction, demonstrating faster convergence and higher accuracy compared to standard convolutional networks.
In our project, we implemented a compact RESNet architecture optimized for the dual-task of predicting policy and value on 19×19 Go boards, under a strict parameter budget of 100,000. The architecture begins with a custom **input stem**, combining 5×5 and 1×1 convolutions in parallel, whose outputs are summed before applying ReLU activation. This hybrid approach captures both global and local spatial information early, without inflating the parameter count. The core of the network consists of **six residual blocks**, each composed of two 3×3 convolutions with batch normalization and ReLU activations, followed by a skip connection. To remain within our parameter constraint, we fixed the number of filters to 25 throughout the network and avoided any expansion or bottleneck layers. All intermediate activations retain full spatial resolution (19×19), as no striding or pooling is used.
The architecture branches into two specialized heads:

- The **policy head** applies a 1×1 convolution with 128 channels, followed by batch normalization, ReLU, and a final 1×1 projection to one channel. After flattening, a softmax produces a probability distribution over the 361 board positions.
- The **value head** uses global average pooling, a dense layer with 50 units and ReLU, and a final sigmoid output to predict the win probability.

The total number of parameters in this RESNet model is **94,042**, of which **93,186** are trainable and **856** are non-trainable, corresponding to approximately **367.35 KB** in memory. This configuration fits within our budget constraint while providing sufficient expressivity for Go board prediction. For training, we used the **adam optimizer**, as it provides stable and adaptive updates, particularly suited for small and noisy gradients commonly encountered in early training epochs. The loss function combines categorical cross-entropy for the policy head and binary cross-entropy for the value head. Evaluation metrics include policy accuracy and mean squared error (MSE) for the value output.
We experimented with two learning rate strategies:

- **Cosine annealing with warm restarts** (Loshchilov et al., 2016), which offers smooth cyclic decay and allows the optimizer to escape shallow local minima. While effective in some configurations, we found it less predictable for long training cycles.
- **A manually defined piecewise schedule**, which we ultimately adopted. This approach gives precise control over learning rate drops at key stages:

$$\text{lr} = \begin{cases} 5 \times 10^{-3}, & \text{if } t < 100 \\ 5 \times 10^{-4}, & \text{if } 100 \leq t < 150 \\ 5 \times 10^{-5}, & \text{if } 150 \leq t < 200 \\ 5 \times 10^{-6}, & \text{if } 200 \leq t < 300 \\ 5 \times 10^{-7}, & \text{if } 300 \leq t < 400 \\ 5 \times 10^{-8}, & \text{otherwise} \end{cases}$$

This staged decay ensured consistent progress throughout the 500 training epochs and allowed for fine-tuning in later stages. Training was conducted with early stopping (patience 20) and

model checkpointing based on validation loss. To ensure robustness, all metrics were logged and saved for later visualization.

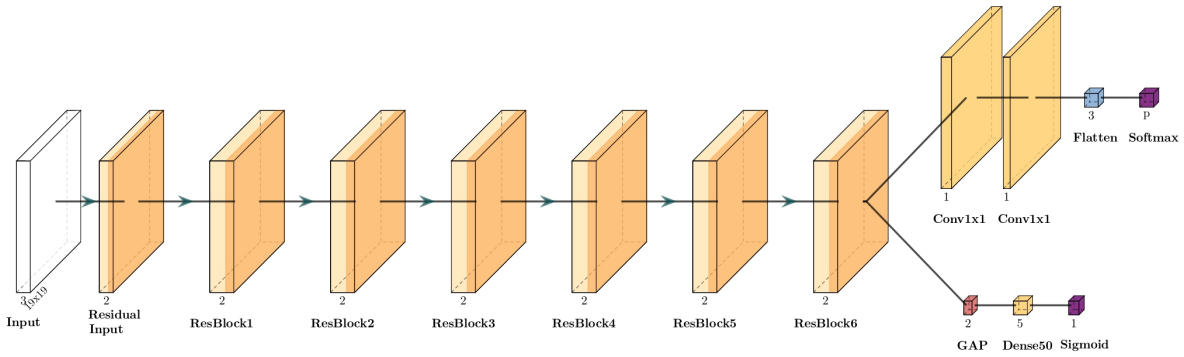

Figure 1: RESNet

3.3.2 MobileNet

MobileNet architectures (Howard et al., 2017; Sandler et al., 2018) were originally developed for mobile vision tasks, but their parameter efficiency makes them highly suitable for constrained environments such as our Go-playing setting. In the context of computer Go, Cazenave (Cazenave, 2020a) showed that MobileNets can rival and sometimes outperform traditional residual networks with far fewer parameters.
In our project, we designed a customized version of MobileNet tailored for the Go board, optimized for dual policy and value prediction under a tight parameter budget. The network starts with a standard 3×3 convolution using 32 filters, followed by **five stacked dual-path blocks**. Each block consists of:

- A 1×1 expansion convolution to 128 channels;
- Two parallel **depthwise separable convolutions**, using 3×3 and 5×5 kernels respectively;
- A concatenation of the two paths, as proposed in (Chen et al., 2017), followed by batch normalization and ReLU;
- A 1×1 projection back to 32 channels, then a **Squeeze-and-Excitation (SE)** block (Hu et al., 2018) to adaptively reweight channel features;
- A residual connection from the input to the block output.

The spatial resolution is preserved throughout the network (no striding or pooling), maintaining the original 19×19 board size. The architecture then splits into two task-specific heads:

- The **policy head** uses a 1×1 convolution with 128 filters, followed by batch normalization, ReLU, a 1×1 projection, flattening, and softmax activation over 361 possible board positions.
- The **value head** applies global average pooling, followed by a dense layer with 50 units and ReLU activation, and a final sigmoid output.

We trained this model for **5,400 epochs** using the **adam optimizer** and a combined loss: categorical cross-entropy for the policy head and binary cross-entropy for the value head. To improve convergence and generalization, we used a **cosine annealing** schedule for the learning rate, defined by the formula:

$$\mathrm{lr}(t) = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{\pi t}{T}\right)\right)$$

where:

- t is the current epoch,
- T is the total number of training epochs,
- $\eta_{\max}$ is the initial learning rate,
- $\eta_{\min}$ is the final learning rate.

The model was trained with `EarlyStopping` (patience 20) and checkpointing based on validation loss.

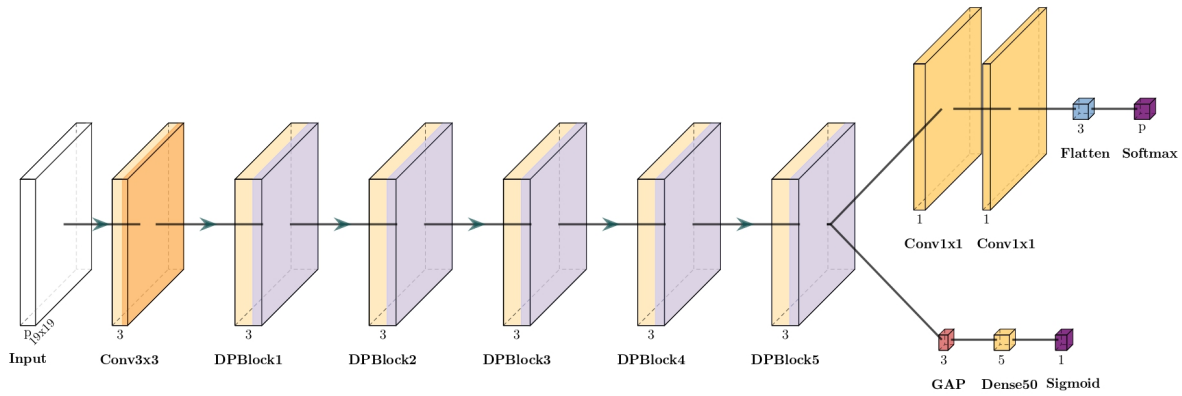

Figure 2: MobileNet

After this initial phase, we extended the architecture by incorporating a **Convolutional Block Attention Module (CBAM)**, which applies sequential channel and spatial attention to enhance feature representation. CBAM has been shown to significantly improve performance across a wide range of visual recognition tasks while maintaining low computational overhead. Inspired by its effectiveness, we integrated CBAM into our Go model to enable more targeted attention to relevant board regions and strategic patterns. The CBAM-enhanced version was fine-tuned for **1,200** additional epochs. The final model contains **107,856** total parameters (including **103,376** trainable and **4,480** non-trainable), slightly exceeding our target of 100,000. This overhead is justified by the inclusion of attention mechanisms, which significantly improve model expressiveness without compromising deployment efficiency. Our findings are consistent with (Cazenave, 2020a), which demonstrated that MobileNet variants, especially those incorporating attention, can outperform similarly-sized residual architectures in Go-specific tasks.

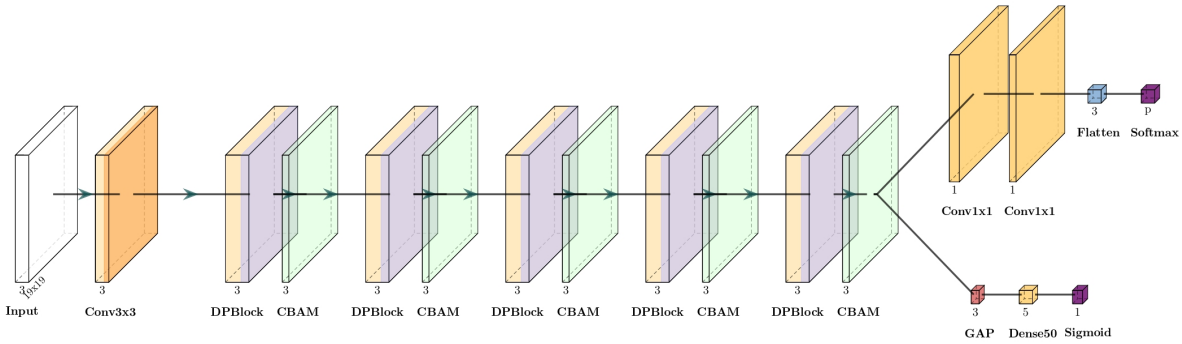

Figure 3: MobileNet + CBAM

3.3.3 Inception

Inception architectures (Szegedy et al., 2016) are known for their ability to capture multi-scale spatial patterns through parallel convolutional paths with varying kernel sizes. Originally designed for large-scale image recognition, these networks have inspired numerous adaptations in resource-constrained environments. Their modularity and efficiency make them well suited for tasks such as Go, where local and global contexts are equally important.
In our project, we implemented a compact Inception-style architecture tailored to 19×19 Go boards, designed under a strict parameter budget of 100,000. Each Inception block is composed of three parallel branches:

- A 1×1 convolution;
- A 3×3 depthwise separable convolution;
- A 5×5 depthwise separable convolution.

These branches are concatenated and projected back to the original number of filters using a 1×1 convolution. To improve feature discrimination, each block includes a **Convolutional Block Attention Module (CBAM)** (Woo et al., 2018), which applies channel and spatial attention sequentially. A skip connection ensures gradient stability and preserves input information. All convolutions use the `swish` activation, and spatial resolution is preserved throughout the network.
The model begins with a 1×1 convolution (41 filters), followed by six inception blocks with CBAM and skip connections. This shared backbone feeds into two output heads:

- The **policy head** uses a 1×1 convolution with 128 filters, followed by batch normalization and another 1×1 projection to a single channel. The output is flattened and passed through a softmax over the 361 board positions.
- The **value head** applies global average pooling, a dense layer with 80 units (swish activation), and a final sigmoid neuron. Dropout (0.3) is added for regularization.

The model was trained for 500 epochs using the **adam optimizer** with an initial learning rate of 10^{-3}. A learning rate scheduler (`ReduceLROnPlateau`) was employed to reduce the learning rate by a factor of 0.5 if validation loss plateaued, with a minimum threshold of 10^{-6}. We also used `EarlyStopping` (patience of 20 epochs) and model checkpointing to retain the best weights.
The final model contains only **94,181** parameters (including **93,925** trainable and **256** non-trainable), occupying less than **368 KB** of memory. Despite its small size, this architecture proved expressive and stable, benefiting from both multi-scale pattern recognition

and lightweight attention. These findings support the claim made in (Woo et al., 2018) that CBAM can be seamlessly integrated into existing backbones to boost performance without significant computational overhead.

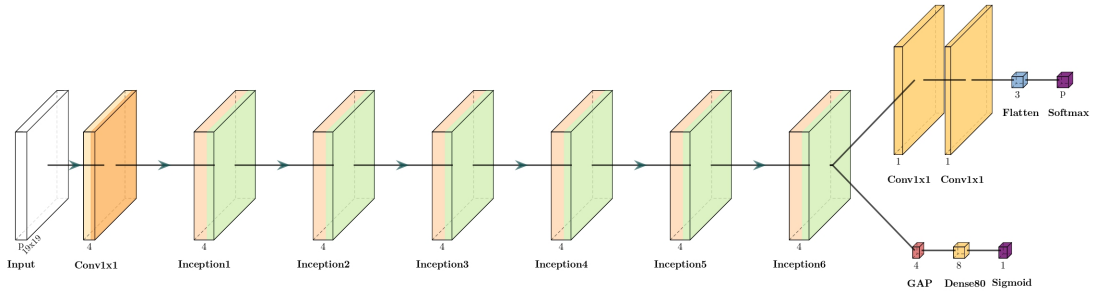

Figure 4: Inception

3.3.4 MNASNet

MNASNet (Tan et al., 2019) is a family of mobile-friendly convolutional architectures discovered via neural architecture search (NAS). Designed to balance accuracy and latency, MNASNet employs **inverted bottleneck blocks**, depthwise separable convolutions, and lightweight expansion layers. Inspired by its structural efficiency, we adapted MNASNet for the task of Go move and value prediction under a 100,000-parameter constraint.
In our implementation, each MNASNet block begins with a pointwise (1×1) convolution that expands the number of channels by a factor of 4, followed by a depthwise convolution with a 3×3 kernel, and finally a projection back to the base number of filters (33). Each block is followed by a combination of two attention mechanisms:

- A **Squeeze-and-Excitation (SE)** block (Hu et al., 2018), which captures global channel context and reweights feature maps accordingly;
- A lightweight **channel attention** mechanism that focuses on the most informative filters via global pooling and gating.

A residual connection is applied when the input and output dimensions match. The network begins with a 3×3 convolution (33 filters), followed by seven MNASNet blocks with attention. The architecture retains full spatial resolution (19×19) throughout.
The two-task output structure includes:

- A **policy head**, composed of a 1×1 convolution with 128 filters, followed by batch normalization, projection, flattening, and a softmax activation over 361 positions;
- A **value head**, using global average pooling, a dense layer with 80 units (ReLU), a dropout layer ($p = 0.3$), and a final sigmoid unit predicting the win probability.

The model was trained over 500 epochs with the **adam optimizer** and a combined loss: categorical cross-entropy for the policy output and binary cross-entropy for the value prediction. To improve generalization and avoid plateaus, we implemented a cosine annealing learning rate schedule with warm restarts. Additional callbacks included `EarlyStopping` (patience = 20), `ModelCheckpoint` to save the best validation model, and `ReduceLROnPlateau` for safety. Metrics and training logs were recorded for post-analysis.
The final model contains approximately **97,897** parameters (of which **93,417** are trainable), consuming less than **383 KB** of memory. The architecture strikes a balance between compactness and expressivity, leveraging compound attention and NAS-inspired structure.

These results highlight the flexibility of MNASNet, especially when enhanced with attention mechanisms for domain-specific tasks like Go.

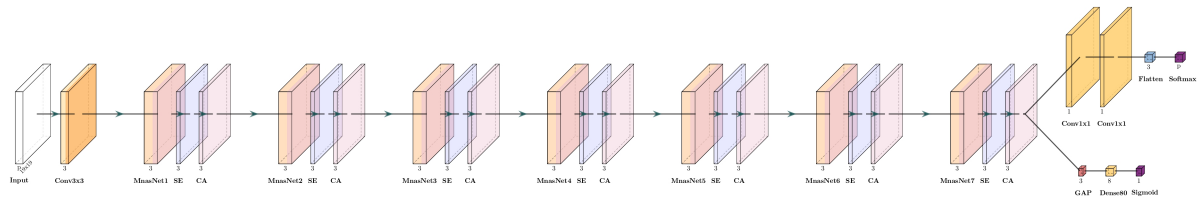

Figure 5: MNASNet

3.3.5 Normalization-Free Networks

Transformer-based architectures have recently gained popularity in vision tasks due to their ability to model long-range dependencies (**linformer**). In the context of board games like Go, spatial patterns extend across the board, and modeling such non-local relationships can be critical. In this experiment, we explored a hybrid model combining a shallow convolutional stem with a multi-layer Transformer encoder stack.
The model begins with a 3×3 convolution using 64 filters to embed the input features, followed by reshaping the $19 \times 19 \times 64$ tensor into a sequence of 361 tokens. These tokens are passed through a stack of **four transformer encoder layers**, each consisting of:

- A multi-head self-attention layer (4 heads, 64 dimensions),
- A feed-forward network with ReLU activation and hidden size of 128,
- A novel non-linearity named **dynamic tanh (DyT)** (Zhu et al., 2025), which replaces normalization layers entirely.

DyT is defined as an element-wise function:

$$\mathrm{DyT}(x) = \gamma \cdot \tanh(\alpha x) + \beta,$$

where α, β, and γ are trainable parameters.

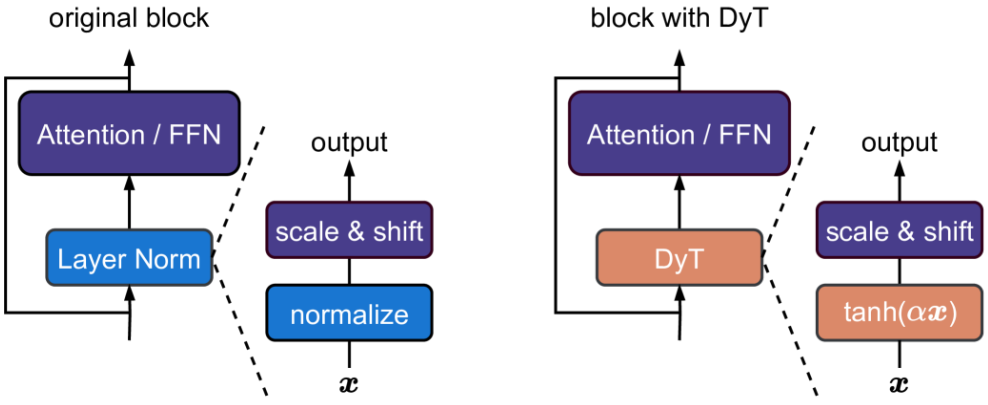

Figure 6: Comparison between a standard Transformer block (left) and a block using the proposed Dynamic Tanh (DyT) layer (right).

This approach was introduced in a recent paper by Zhu et al. (Zhu et al., 2025), which challenges the widespread assumption that normalization layers are essential to modern neural network

training. The authors demonstrate that DyT can match or surpass traditional normalization strategies across vision and language tasks, while simplifying architecture design and reducing computational overhead.
In our implementation, DyT replaces both layer normalization and activation layers, offering a compact and theoretically grounded alternative. The Transformer outputs feed into two heads:

- A **policy head**, applied to the first token in the sequence, producing a probability distribution over the 361 board positions;
- A **value head**, built from global average pooling, a 50-unit dense layer, and a final sigmoid output.

The model was trained using the Adam optimizer for 500 epochs. However, it contains approximately **377,494 parameters**, which is far beyond our constraint of 100,000. Due to the architectural complexity and time constraints, we were not able to optimize this model further.
Note: Due to technical limitations, the DyT module originally proposed in PyTorch was not fully compatible with TensorFlow. Some dynamic behaviors of the DyT operator led to runtime issues in our training pipeline. As a result, we could not fully validate this model in practice, and it remains a theoretical exploration in our project. Despite these challenges, this experiment illustrates the promise of hybrid token-based architectures for board games, and the growing relevance of normalization-free Transformer variants like DyT.

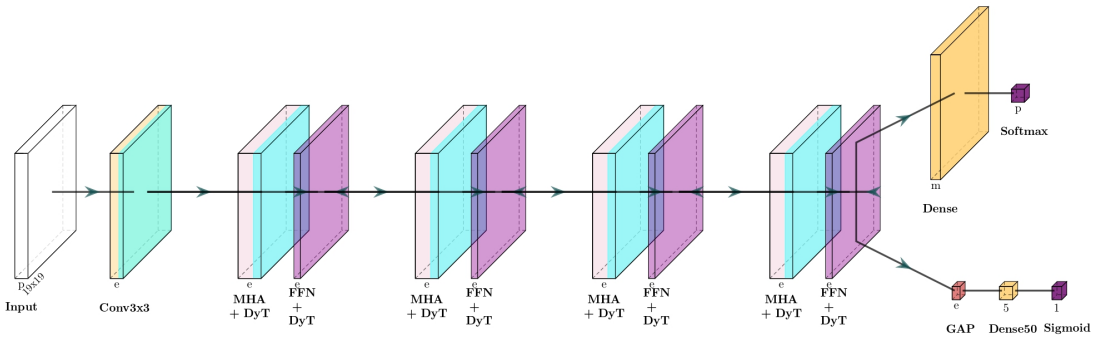

Figure 7: Normalization-Free Networks

3.3.6 Hybrid CNN + Linformer

To explore attention-based mechanisms while remaining within a strict parameter budget, we implemented a hybrid model combining lightweight convolutions with linear self-attention. This architecture draws inspiration from the Linformer (Wang et al., 2020), a transformer variant designed for efficient sequence modeling through low-rank attention projection. In our setting, it was adapted to operate on spatial Go board representations without exceeding 100,000 parameters.
The model begins with a 3×3 convolution followed by batch normalization and ReLU, producing a 32-channel feature map. This is followed by a stack of four **residual convolutional blocks**, each composed of two 3×3 convolutions and batch normalization layers. Each block is augmented with a **Squeeze-and-Excitation (SE)** mechanism (Hu et al., 2018) to enhance channel-wise interactions, while maintaining spatial resolution throughout the network.
The output of the CNN stack is reshaped and passed to a compact **Linformer block**, where the 19×19 spatial grid is flattened to a 361-token sequence. Multi-head self-attention is applied with low-rank projections, and residual connections are added around both the attention and

feed-forward sublayers, each followed by layer normalization. This structure allows the model to attend globally across the board while remaining efficient and lightweight.
The model splits into two output heads:

- The **policy head** uses two 1×1 convolutions, the first with 32 filters and ReLU, the second projecting to a single channel. The output is flattened and passed through a softmax over the 361 board positions.
- The **value head** applies global average pooling followed by a dense layer with 80 units (ReLU), dropout for regularization, and a final sigmoid output to predict the win probability.

Training was performed over 500 epochs using the Adam optimizer and a loss combination of categorical cross-entropy (policy) and binary cross-entropy (value). We used a cosine annealing learning rate schedule with warm restarts, implemented via a custom callback. The final model contains **94,826** parameters (370 KB), making it a strong candidate under the 100k budget. While compact, it integrates both local and global reasoning capabilities, leveraging SE blocks and efficient attention.

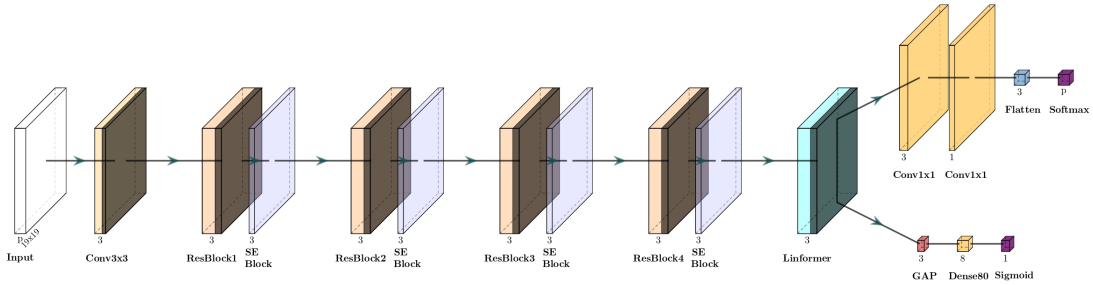

Figure 8: Hybrid CNN + Linformer

3.3.7 EfficientNetFormer

Inspired by the search for efficient hybrid models blending convolution and attention, **EfficientFormer** (Li et al., 2022) introduces a scalable design leveraging lightweight depthwise convolutions, channel attention, and a form of linear attention. Originally developed for mobile visual recognition tasks, this architecture has shown promising results in low-complexity settings, making it a compelling candidate for Go policy and value prediction under strict parameter constraints.
In our adaptation, we designed a compact version of EfficientFormer tailored to the 19×19 Go board input and constrained to stay within 100,000 parameters. The architecture starts with a standard 3×3 convolution layer followed by **five EfficientFormer blocks**. Each block includes:

- A 1×1 expansion convolution (increasing channels by a factor of 3);
- A depthwise 3×3 convolution for spatial feature extraction;
- A lightweight **linear attention mechanism**, implemented as a global average pooling followed by a channel-wise excitation (gating) to modulate the feature maps;
- A 1×1 projection layer to reduce the feature dimension;
- A residual connection (if dimensions match).

The policy and value heads follow the standard dual-head architecture defined across our models:

- The **policy head** includes a 1×1 convolution with 64 channels, batch normalization, a final 1×1 projection, and a softmax over the flattened output.
- The **value head** uses global average pooling, followed by a dense layer with 40 hidden units, dropout, and a sigmoid output for win probability.

We trained the model using the **adam optimizer** and a **cosine annealing learning rate schedule with warm restarts**, implemented via a custom callback. This strategy allowed progressive decay and recovery of the learning rate across training epochs, encouraging exploration of multiple local minima. Training was monitored with `EarlyStopping` (patience of 20) and validation-based checkpointing.
The final model includes a total of **98,866** parameters (376.7 KB trainable), successfully respecting the project constraint while leveraging the EfficientFormer design to incorporate both convolutional inductive bias and lightweight attention mechanisms.

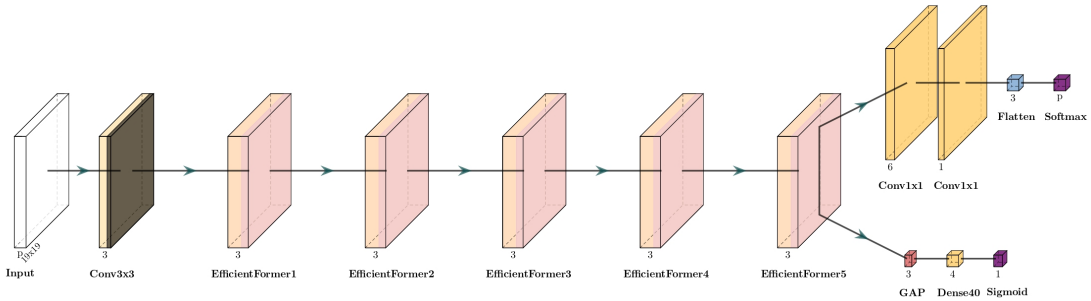

Figure 9: EfficientFormer

4 Experiments and Results

In this section, we evaluate and compare the performance of the different lightweight architectures developed for the game of Go. All models were trained on a dataset of 1,000,000 self-played games generated by KataGo, with a fixed input shape of $19 \times 19 \times 31$ planes. Each model respects the constraint of approximately 100,000 parameters and follows a dual-head structure with a policy output (categorical cross-entropy) and a value output (binary cross-entropy). Training was conducted over 500 epochs (up to 5400 for MobileNet), using the Adam optimizer and either piecewise or cosine annealing learning rate schedules. Regularization was applied via L2 and dropout, and early stopping was enabled to prevent overfitting. Every 20 epochs, the model is evaluated on 10,000 synthetic positions generated via `golois.getValidation()`, providing an estimation of generalization performance.

Among all tested architectures, **MobileNet** emerged as one of the most promising models, demonstrating competitive performance and rapid convergence despite its compact size. While all models were initially trained for 500 epochs, we extended the training of MobileNet to **5400** epochs in order to explore its long-term learning capabilities. This decision was based on the observation that MobileNet's architecture, built upon depthwise separable convolutions and residual connections, showed no signs of overfitting during early training stages. Moreover, the inclusion of Squeeze-and-Excitation (SE) blocks made it highly expressive despite its limited parameter count, motivating the hypothesis that it could continue to benefit from extended optimization when paired with appropriate regularization and learning rate decay.

The extended training resulted in consistent improvements across all metrics. The final validation loss reached **2.5196**, with the policy and value heads contributing **1.9281** and **0.5914**

respectively. The model achieved a policy accuracy of **0.4660** on unseen positions, and a value prediction mean squared error (MSE) of **0.0772**, outperforming most other architectures evaluated under the same conditions. These results confirm that MobileNet is well suited to the Go domain, efficiently capturing spatial dependencies and move patterns while maintaining generalization performance.

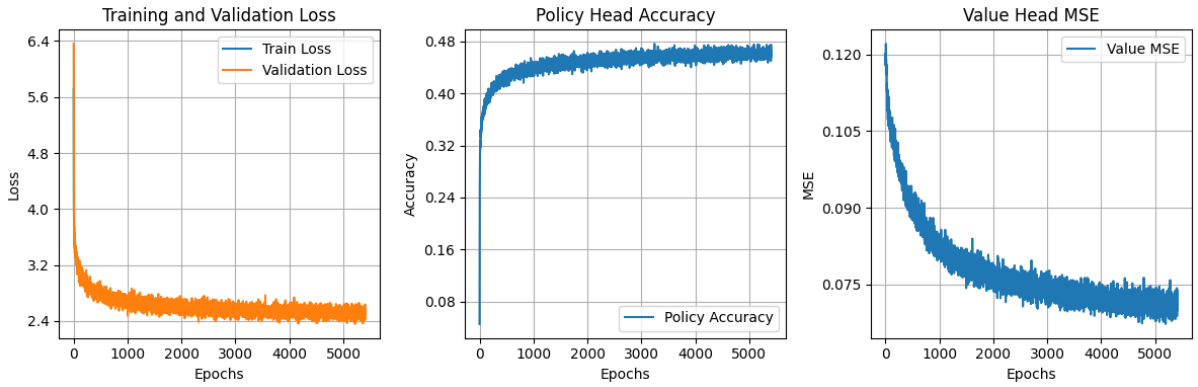

Figure 10: Training curves for MobileNet trained over 5400 epochs. Left: training and validation loss. Center: policy head accuracy. Right: value head mean squared error (MSE).

To further improve the expressiveness of the baseline MobileNet, we integrated a Convolutional Block Attention Module (CBAM) into each residual block. CBAM applies lightweight channel and spatial attention mechanisms sequentially, allowing the network to dynamically focus on the most relevant features in the input space. This integration adds minimal overhead in terms of parameters but significantly boosts the model's ability to capture strategic Go patterns across the board. The CBAM-enhanced MobileNet was trained for 1200 epochs, following a training schedule similar to the baseline. The extended training allowed the model to fully converge and leverage the attention mechanisms effectively. The final validation results demonstrated consistent improvements: the total loss decreased to **2.4362**, including a policy loss of **1.8737** and a value loss of **0.5626**. The policy accuracy reached **0.4744**, the highest among all models tested, while the value head achieved a mean squared error of only **0.0664**. These improvements confirm that attention mechanisms even in compact form can substantially enhance the performance of lightweight Go models by promoting focus on key regions and reducing noise from less relevant inputs.
The performance gains obtained through CBAM demonstrate that architectural enhancements focused on attention and dynamic feature reweighting can be applied successfully within strict parameter budgets. Despite a slight increase in parameter count (approximately 107k), the CBAM-MobileNet remains deployable in constrained environments while delivering state-of-the-art results within this benchmark.

Beyond MobileNet, we evaluated a diverse set of compact architectures including **EfficientNet**, **Inception + CBAM**, **MNASNet**, **ResNet**, and a **Hybrid CNN + Linformer**. Each model was trained for 500 epochs under identical conditions, using the same loss functions, regularization, and learning rate schedules. All models respect the 100k parameter constraint and implement a dual-head output structure.
The following figures summarize the training and validation performance of these architectures:

- **Figure 11**: Policy Head Accuracy across epochs.
- **Figure 12**: Total Loss (Policy + Value) over training.
- **Figure 13**: Value Head Mean Squared Error (MSE).

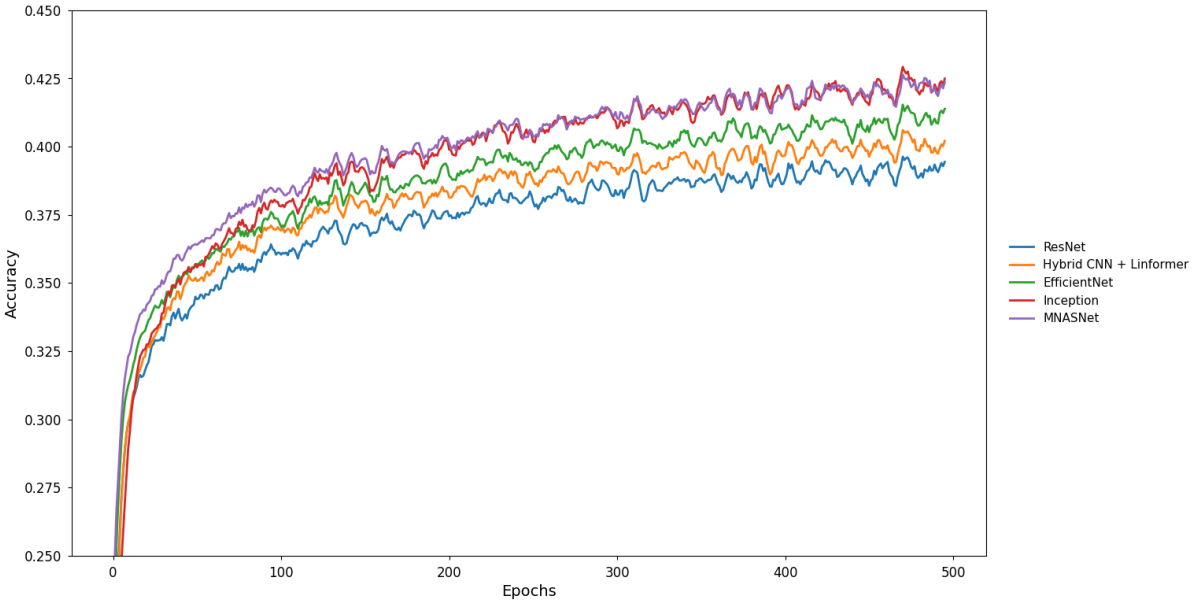

Figure 11: Validation accuracy of the policy head across training epochs for all tested architectures.

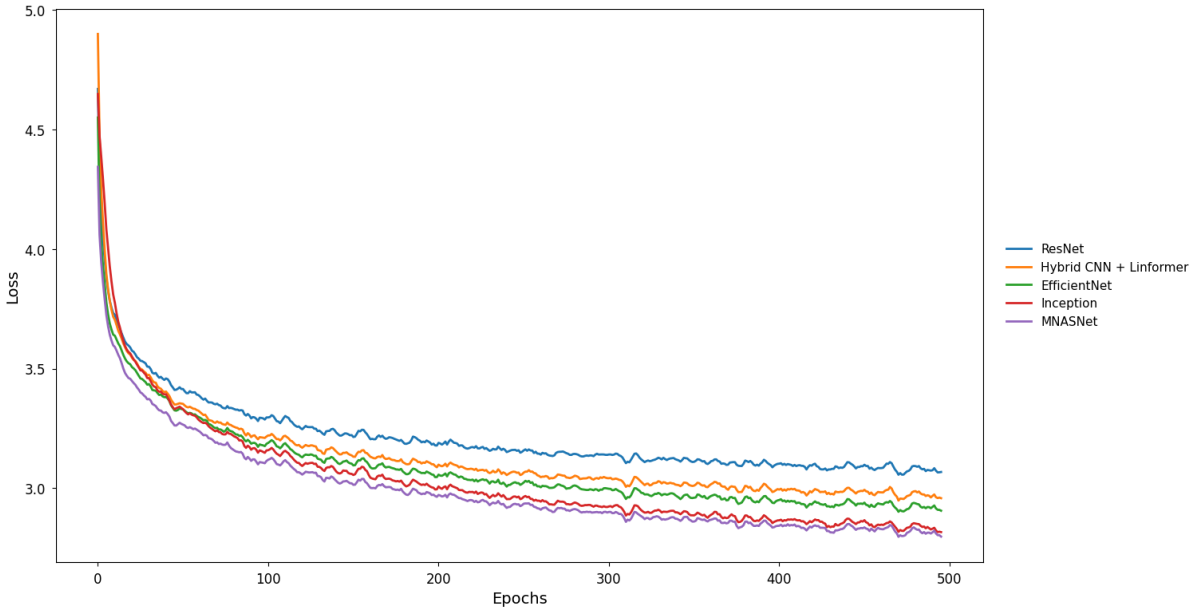

Figure 12: Validation loss curves over 500 epochs for each model.

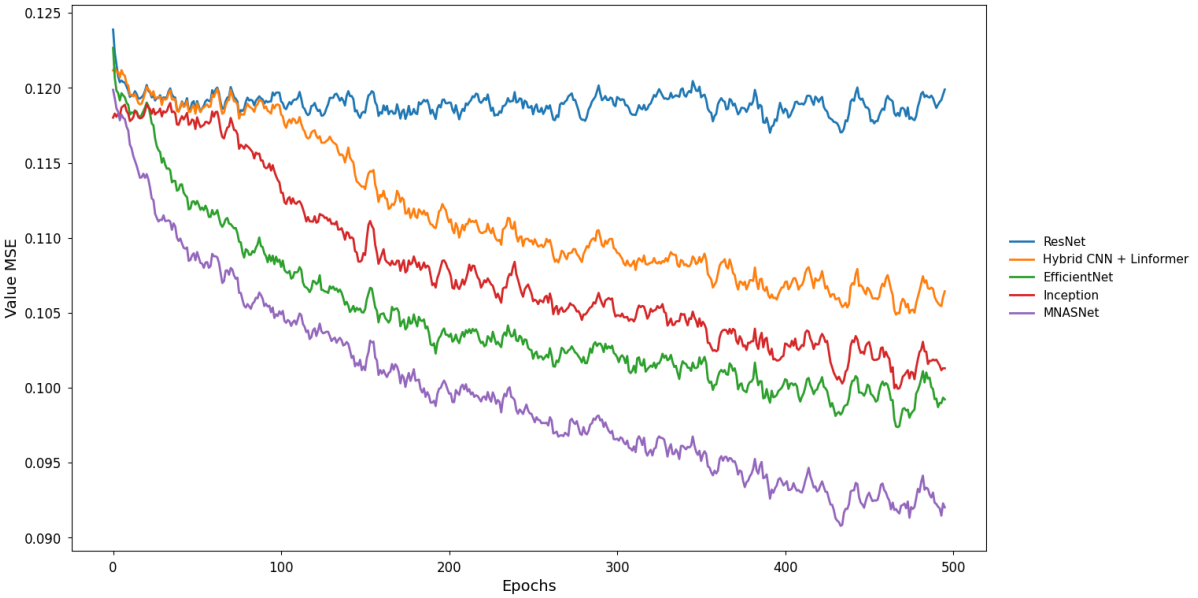

Figure 13: Validation MSE of the value head during training. Lower values indicate better win probability prediction.

Table 1 presents a summary of the best validation metrics achieved during training:

Model	Policy Accuracy	Value MSE	Final Loss
ResNet	0.4032	0.1153	3.0306
Hybrid CNN + Linformer	0.4126	0.1035	2.9243
EfficientNet	0.4217	0.0952	2.8846
Inception + CBAM	0.4340	0.0973	2.7816
MNASNet + Attention	0.4331	**0.0897**	**2.7672**

Table 1: Performance summary of all architectures over 500 epochs.

Among the convolution-based models, **Inception + CBAM** and **MNASNet** emerged as the most performant, achieving the highest policy accuracy and lowest validation loss respectively. EfficientNet also showed solid generalization, benefiting from its compound scaling and depthwise convolutions. In contrast, the **ResNet** model, while stable and consistent, lagged behind in accuracy and value estimation performance. This suggests that the absence of attention mechanisms and multi-scale processing may limit its expressiveness under a strict parameter budget. The **Hybrid CNN + Linformer** achieved a good balance between local and global reasoning, with moderately low loss and MSE. These results confirm that architectural innovations such as multi-branch convolutions and attention modules (e.g., CBAM, SE) contribute significantly to lightweight model efficiency in the game of Go.
To complement offline validation metrics, all models were evaluated in a round-robin tournament organized weekly by the professor. In the final stage, each architecture played 168 games against others using a fixed lightweight Monte Carlo Tree Search (MCTS). This setup allowed us to measure each model's actual playing strength through win rate and consistency (σ).It is important to note that the **Inception + CBAM** model, despite achieving the highest validation policy accuracy **(0.4340)**, could not be evaluated in the tournament. This was due to a TensorFlow deserialization error caused by the CBAM layer not being recognized during model loading. Table 2 presents a summary of the final evaluation of all models. It includes key validation metrics (policy accuracy and value MSE), the number of parameters, as well as the

results from the round-robin tournament: win rate and its standard deviation σ, which reflects the consistency of each model across games.

Model	Parameters	Policy Acc	Value MSE	Win Rate (%)	σ
ResNet	94,042	0.4032	0.1153	18.5	0.030
Hybrid CNN + Linformer	94,826	0.4126	0.1035	29.8	0.035
EfficientNetFormer	98,866	0.4217	0.0952	25.6	0.034
Inception + CBAM	94,181	**0.4340**	0.0973	—	—
MNASNet + Attention	97,897	0.4331	**0.0897**	35.7	0.037
MobileNet + CBAM	107,856	**0.4744**	**0.0664**	**83.3**	**0.029**

Table 2: Final tournament results: parameters, validation metrics, win rates and standard deviation

As observed, the **CBAM-enhanced MobileNet** and long-trained **MobileNet** models both achieved the best results with a win rate of **83.3%**, confirming their robustness and generalization capacity in actual gameplay. Their validation metrics (policy accuracy $> 46\%$, MSE < 0.08) are also the best in this benchmark.

The **MNASNet** model showed strong generalization during training with the lowest value head MSE (0.0897), but this did not fully reflect in its gameplay performance. Similarly, **Inception + CBAM** stood out with the highest policy accuracy (0.4340) and a very low training loss, confirming the benefits of combining multi-scale convolution and attention. Other models like **EfficientNetFormer** and **Hybrid CNN + Linformer** displayed moderate success in both metrics and games, while the **ResNet** baseline remained behind due to its lack of attention or multi-scale capacity.

The ultimate goal was to outperform the professor's reference model `se.2022.10.20.Adam.binary.cosine.student.504.0.002.1.0.120` (91,741 parameters), which achieved an exceptional **win rate of 95.8%** over 168 games (161 wins, $\sigma = 0.015$). While our models did not surpass this baseline, the MobileNet variants narrowed the gap significantly under tight parameter constraints.

These results confirm that attention-enhanced lightweight models such as CBAM-MobileNet offer strong real-world playing performance, narrowing the gap with high-performance baselines, while also maintaining compactness and interpretability.

5 Discussion

This study enabled an in-depth evaluation of several compact architectures for the game of Go, all strictly constrained to fewer than **100,000** parameters. Among them, MobileNet variants enhanced with attention modules (SE, CBAM) stood out by achieving a tournament win rate of **83.3%**, a policy accuracy exceeding **47%**, and a particularly low mean squared error (MSE) of approximately **0.066**. However, these results must be interpreted with caution, as they were obtained through extended training over 5400 epochs, while all other models were limited to 500 epochs due to insufficient computational resources. This highlights the critical role of training time: under strong capacity constraints, a longer training schedule can significantly boost model expressiveness and generalization, becoming as crucial as architectural design itself. Architectures such as EfficientFormer and Linformer, although promising from a structural standpoint, were not given the opportunity to reach their full potential, as suggested by their still-decreasing loss curves at epoch 500. Likewise, Normalization-Free Networks and token-based models like Dynamic Tanh Transformers could not be fully tested due to library compatibility issues, especially when training on Google Colab with TensorFlow. These

constraints, although external to model design, remain important considerations in lightweight AI development. A key observation from this study concerns the asymmetry in optimization behavior between the network's two output heads. The value head, solving a binary classification task, converged more rapidly and reliably across all models. In contrast, the policy head, which outputs a probability distribution over 361 board positions, proved harder to optimize, showing slower and more unstable learning. This difference underlines the importance of designing architectures that enhance spatial discrimination—such as those employing attention mechanisms or multi-kernel convolutional paths—so as to improve policy accuracy, which is ultimately critical for gameplay performance. Another promising direction for future research lies in the integration of emerging paradigms such as Kolmogorov–Arnold Networks (KANs), which propose a novel approach to function approximation through sparse and interpretable structures (Zhu et al., 2025). Although these networks were not used in this study, their theoretical efficiency and explainability make them appealing candidates for designing Go models under strict parameter constraints. Finally, while our models were trained in a purely supervised manner, future work could benefit from knowledge distillation techniques (Hinton et al., 2015), transferring strategic insight from larger teacher networks into smaller student models. In particular, recent work such as **Rapfi** (Jin et al., 2024) demonstrates the effectiveness of distillation in board games like Gomoku, where compact student networks learned from larger CNNs can surpass traditional agents even under severe computational limitations. This approach could help bridge the gap between compact architecture constraints and the strategic depth required for high-level play, without relying exclusively on reinforcement learning.

6 Conclusion and future work

In this work, the design of lightweight neural network architectures for the game of Go was explored under the strict constraint of 100,000 parameters. Through an extensive evaluation of classical convolutional networks and modern attention-based or hybrid architectures, it was demonstrated that compact models, especially MobileNet variants with attention modules, can achieve strong playing performance when combined with careful design and sufficient training time. The results highlight key trade-offs between model capacity, training duration, and architectural efficiency, particularly regarding the differential convergence behavior of the policy and value heads. Future work will focus on extending the evaluation to architectures that were promising but not fully explored due to computational limits, such as normalization-free Transformers and models incorporating Dynamic Tanh (DyT). Additionally, the potential of Kolmogorov–Arnold Networks (KANs), which offer a new paradigm for function approximation with fewer inductive biases, will be assessed. Knowledge distillation techniques could also be applied to transfer knowledge from larger models to the compact architectures, improving efficiency without increasing parameter count. These approaches , combined with improved training schedules and hybrid architectural designs, represent promising directions for advancing lightweight Go-playing agents.

References

Cazenave, T. (2020a). Mobile networks for computer go. *arXiv preprint arXiv:2008.10080*.

Cazenave, T. (2020b). Residual networks for computer go. *IEEE Transactions on Games*, *12*(1), 3–10.

Cazenave, T. (2021). Improving generalized puct for computer go. *arXiv preprint arXiv:2008.10080*.

Cazenave, T., Sentuc, J., & Videau, M. (2021). Cosine annealing, mixnet and swish activation for computer go. *arXiv preprint arXiv:2102.03467*.

Chen, Y., Li, J., Yu, H., Chen, S., Loh, P. C., & Yan, J. (2017). Dual path networks. *Advances in Neural Information Processing Systems (NeurIPS)*, *30*.

Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. *Computers and Games*, 72–83.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks, 7132–7141.

Jin, Z., Duan, H., Hang, Z., & Xu, C. (2024). Rapfi: Distilling efficient neural network for the game of gomoku [Withdrawn ICLR 2025 Submission]. *arXiv preprint arXiv:2409.12345*.

Li, Z., Xu, H., & Liu, S. (2022). Efficientformer: Vision transformers at mobilenet speed. *arXiv preprint arXiv:2206.01191*.

Loshchilov et al. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Pascutto, G.-C. (2017). Leela zero.

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Sagri, A., Cazenave, T., Arjonilla, J., & Saffidine, A. (2023). Vision transformers for computer go. *arXiv preprint arXiv:2309.12675*.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C., et al. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. *CVPR*, 4510–4520.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 1–9.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2818–2826.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile, 2820–2828.

Tian, Y., Gong, Q., Keutzer, K., et al. (2019). Mastering the game of go without human knowledge: Opengo and elf. *arXiv preprint arXiv:1902.04522*.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., & Ma, H. (2020). Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Woo, S., Park, J., Lee, J.-Y., & Kweon, I. S. (2018). Cbam: Convolutional block attention module. *Proceedings of the European Conference on Computer Vision (ECCV)*, 3–19.

Wu, D. J., et al. (2019). Accelerating self-play learning in go. *arXiv preprint arXiv:1902.10565*.

Wu, D. J. (2021). Katago.

Zhu, J., Chen, X., He, K., LeCun, Y., & Liu, Z. (2025). Transformers without normalization. *arXiv preprint arXiv:2503.10622*.