1. A brief summary of the paper's goals or techniques introduced (if any).
   * Due to the shallow search techniques from tree search they were unable to tackle problems such as Go efficiently
   * Used CNN to reduce depth and breath of search tree
   * Using a value network
   * Sampling action using a policy network
   * Most challenging of classic games for artificial intelligence
   * Enormous search space and the difficulty of evaluating board positions and moves
   * New search algorithm that combines monte carlo simulation with value and policy networks
   * Exhaustive search is infeasible
   * Depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function that predicts the outcomes from state s
   * Breadth of the search may be reduced by sampling actions from a policy that is a probability distribution over possible moves a in position s
   * Limited to shallow
   * Employ a similar architecture for the game of Go
   * Uses convolutional layers to construct a representation of the position
   * Se these NN to reduce the effective depth and breadth of the search tree: evaluating positions using a value network, and sampling actions using a policy network
   * Predict human expert moves in a dataset of positions
   * Using stochastic gradient ascent to maximize the likelihood of the human move
   * 13 layer policy network
   * select an action rather than 3ms for the policy network
   * improving the policy network by policy gradient reinforcement learning(RL)
   * use a reward function r(s) that is zero for all non-terminal time steps t<T.
   * The outcomes is the terminal reward a the end of the game form the perspective of the current player at timestep t: +1 for winning and -1 for losing. Weights are then updated at each time step t by stochastic gradient ascent in the direction that maximizes expected outcome
   * Each simulation tranverse the tree by selecting the edge with maximum action value Q. plus a bonus u(P) that depends on a stored prior probability P for that edge
   * Position evaluation, estimating a value function v(s) that predicts the outcome from position s of games played using policy p for both players
   * We would like to know the optimal value function under perfect play
   * Estimate the value function fro the strongest policy, using the RL policy network
   * The value network memorized the game outcomes rather than generalising to new positions
   * Single evaluation of v(s) also approached the accuracy of Monte Carlo rollouts using the RL policy network but using 15,000 times less compution
   * RL optimizes for the single best move
   * To efficiently combine MCTS with deep neutral networks, ALphaGO uses an asynchronous multi-threaded search that executes simulations on CPus, and computes policy and value networks in parallel on GPUs
   * Search algorithm:
   * Implemented an asynchronous policy and value MCTS algorithm
   * Search control strategy initially prefers actions with high prior probability and low visit count, but asymptotically prefers actions with high action value
   * This virtual loss discourages other threads from simultaneously exploring the identical variation
   * Rollout policy p(a|s) is a linear softmax policy based on fast, incrementally computed, local pattern-based features consisting of both ‘response’ patterns around the previous move that led to state s, and ‘non-response’ patterns around the candidate move a in state s
   * Symmetries of Go have been exploited by using rotationally and reflectionally invariant filters in the convolutional layers
   * We exploit symmetries at run-time by dynamically transforming each position s using the dihedral group of eight reflections and rotations
   * 3 week training time for 340 million training steps
   * policy network was trained using 10,000 minibatches for 128 games on 50 GPUs
   * used nosier distributions as to increase the diversity of the dataset
   * features used were raw representation of the game rules
   * indicating the status of each intersection of the Go board
   * Neutral network architecture
     1. 19x19x48 image stack
     2. 48 feature planes
     3. hidden layer zero pads input into a 23 x 23 image
     4. convolves k filters of kernel size 5x5 with stride 1 with the input image and applies a rectifier non-linearity
     5. hidden layers 2 -> 12 -> 21x 21 image -> 3x3 kernel size with stride of 1 -> relu
     6. input
        1. 19x19 x 48 image stack
        2. binary feature plan describing the current colour to play
        3. Hidden layers 2-11 + additional CNN layer as 12th layer
        4. + 13th layer using hiddent to convole to 1x1
        5. + 14th FC layer with 256 retifiler units
        6. output using single tanh unit
2. A brief summary of the paper's results (if any).
   * Achieved 99.8% winning rate against other go programs