APPENDIX

A PROOFS

A.1 PROOF OF PROPERTY 3

Proof. Constraint (6) taking $k = 0$ is equivalent to Constraint (3), ensuring that $z_0 \in \mathcal{B}_\epsilon(x)$. Thus, $v(QP_t^j) \geq 0$ implies $\min_{z_0 \in \mathcal{B}_\epsilon(x)} z_K^y - z_K^j \geq 0$. □

A.2 PROOF OF THEOREM 1

Proof.

1. We first prove that $v(QP_t^{\bar{j}}) \geq v(QP_u)$ for all $\bar{j} \in \bar{\mathcal{J}}_K$. Let z^* be an optimal solution of $(QP_t^{\bar{j}})$. We build the solution $(z = z^*, \beta)$ of (QP_u) with $\beta_{\bar{j}} = 1$ and $\beta_j = 0 \quad \forall j \in \bar{\mathcal{J}}_K \backslash \{\bar{j}\}$ satisfying Constraints (19) and (18). Obviously, the two objective functions have the same value.
2. Then, we prove that $\min_{\bar{j} \in \bar{\mathcal{J}}_K} v(QP_t^{\bar{j}}) \leq v(QP_u)$.
   Let (z^*, β^*) an optimal solution of (QP_u) with $\beta_{\bar{j}} = 1$ for $\bar{j} \in \bar{\mathcal{J}}_K$. Obviously, $z = z^*$ is feasible for $(QP_t^{\bar{j}})$. Once again, the objective values are the same.

□

A.3 PROOF OF PROPOSITION 1

Proof. We ensure that the inequalities are valid in all three possible cases:

1. If $\beta_{j_1} = 1$ and $\beta_{j_2} = 0$, then both sides of (26) are equal to 0 and (27) becomes $U^{j_1} \geq z_K^{j_1}(x)$.
2. $\beta_{j_1} = 0$ and $\beta_{j_2} = 1$, (26) and (27) respectively lead to $z_K^{j_2}(x) \leq U_K^{j_2}$ and $z_K^{j_2}(x) \geq z_K^{j_1}(x)$. The latter is valid since when $\beta_{j_2} = 1$, class j_2 provides the worst example.
3. Otherwise, $\beta_{j_1} = \beta_{j_2} = 0$ and we obtain $L_K^{j_1} \leq z_K^{j_1}(x)$ from (26) and $U_K^{j_1} \geq z_K^{j_1}(x)$ from (27).

□

A.4 CONSTRAINTS EQ. (13) OR EQ. (9) ALSO IMPLIES $L_k \leq P[z_k] \leq U_k$ IN THE SDP CASE

Proof. We want to prove that $P[z_k z_k^T] - (U_k + L_k)P[z_k] + U_k L_k \leq 0$ implies $L_k \leq z_k \leq U_k$. Assume P is positive semidefinite, so every principal minor (ie. every determinant of any principal submatrix) is nonnegative. In particular : $det \begin{pmatrix} P[1] & P[z_k] \\ P[z_k] & P[z_k z_k^T] \end{pmatrix} \geq 0$, and since $P[1] = 1$, this gives $P[z_k]^2 \leq P[z_k z_k^T]$. Plugging $P[z_k]^2$ into Eq. (13) we obtain $P[z_k]^2 - (U_k + L_k)P[z_k] + U_k L_k \leq 0$, which is equivalent to $(U_k - P[z_k])(P[z_k] - L_k) \geq 0$. $L_k \leq U_k$ gives : $L_k \leq P[z_k] \leq U_k$.

The same reasoning applies with P_k. □

B COMPLEMENTS ON METHOD

B.1 ALGORITHMS COMPARAISON

$\texttt{SDP}_t\ (\epsilon, x)$
cert $\leftarrow$ true;
for $j \in \bar{\mathcal{J}}_K$ **do**
  Compute bounds with β-CROWN.
  if $\exists l \in \bar{\mathcal{J}}_K \cup y,\ \ U_K^j < L_K^l$ **then**
    break;
  $v_j^* \leftarrow$ Solve $(SDP_t^{\epsilon,j,x})$;
  if $v_j^* \leq 0$ **then**
    cert $\leftarrow$ false;
    break;
return cert;

Algorithm 1: Full certification of ϵ-robustness of a DNN with targeted SDP model $(SDPt)$ on data $x \in \mathcal{X}$

$\texttt{SDP}_u\ (\epsilon, x)$
cert $\leftarrow$ true;
Compute bounds with β-CROWN.
T $\leftarrow \{j \in \bar{\mathcal{J}}_K, \quad \exists l \in \bar{\mathcal{J}}_K \cup y, \quad U_K^j < L_K^l\}$
$v^* \leftarrow$ Solve $(SDP_u^{\epsilon,T,x})$;
if $v^* \leq 0$ **then**
  cert $\leftarrow$ false;
  break;
return cert;

Algorithm 2: Full Certification of ϵ-robustness of a DNN with untargeted SDP model (SDP_u) on data $x \in \mathcal{X}$

B.2 STABLE ACTIVE NEURONS ABLATION

In this section, we explain with more details our ReLU constraint relaxation in the context of stable active neurons ablation. The quadratic ReLU constraint on neuron j of layer $k+1$ is

$$z_{k+1}^j(z_{k+1}^j - W_{k+1,u}^j z_k^u - b_k) = z_{k+1}^j W_{k+1,a}^j z_k^a$$

where z_k^a is the subset of stable active neurons of layer k. Note that each stable active neuron r of layer k can be decomposed into a linear combinations of previous layers outputs $z_k^r = \sum_{l=0}^{k-1} \sum_{i=1, i \text{ unstable}}^{n_l} \lambda_l^i\ z_l^i\ +\ \gamma$, where $\gamma \in \mathbb{R}, \lambda_l^i \in \mathbb{R} \quad \forall l \in [0, k-1], i \in [1, n_l]$. This decomposition can be computed dynamically :

- for a layer $k = 1$ and a stable active neuron r on such layer: $z_1^i = W_1^i z_0 + b_1^i$;
- for a layer $l > 1$ and a stable active neuron r on such layer,

$$\begin{aligned} z_k^r &= W_k^r z_{l-1} + b_k^r \\ &= W_{k,u}^r z_{k-1}^u + W_{k,a}^r z_{k-1}^a + b_k^r \\ &= W_{k,u}^r z_{k-1}^u + \sum_{l=0}^{k-2} \sum_{i=1, i \text{ unstable}}^{n_l} \Big(\sum_{\nu=1, \nu \text{ active}}^{n_{k-1}} W_{k,\nu}^r\ \lambda_{l,\nu}^i \Big) z_l^i + \sum_{\nu=1, \nu \text{ active}}^{n_{k-1}} \gamma_\nu + b_l^r \end{aligned}$$

;

Finally, our ReLU quadratic constraint can be rewritten

$$z_{k+1}^j(z_{k+1}^j - W_{k,u}^j z_k^u - b_k) = \sum_{l=0}^{k-1}\sum_{i=1}^{n_l} A_l^i z_{k+1}^j z_l^i + B z_{k+1}^j \tag{32}$$

where A is derived from the product of W_{k+1}^a and the coefficients λ, and B is derived from the product of W_{k+1}^a and the coefficients γ.

We now consider the quadratic term $z_{k+1}^j z_l^i$ present in the non-relaxed ReLU constraint on neuron j of layer $k+1$, where l is a previous layer $l \in [0, k-1]$ and i is a neuron on such layer. Due to the chordal decomposition, only products of neurons of two consecutive layers are considered and the quadratic terms $z_{k+1}^j z_l^i$ do not appear in our matrix variables. Our idea is to bound the terms $z_{k+1}^j z_l^i$ using the upper and ower bounds given by McCormick envelopes McCormick (1976). We have the following bounding inequalities $0 \leq z_{k+1}^j \leq U_{k+1}^j$, $\tilde{L}_l^i \leq z_l^i \leq U_l^i$, where:

- U_{k+1}^j and U_l^i are computed with β−CROWN.
- $\tilde{L}_l^i$ are computed as follows:
  - For the input $l = 0$, we take the known trivial lower bounds : $\tilde{L}_l^i = L_0^i = x^i - \epsilon$;
  - For hidden layers $l > 0$, as all stable neurons have been pruned, neuron i of layer l is unstable, that is to say its lower bound computed by β−CROWN is negative : $L_l^i < 0$. As z_l^i represents the post-activation value and we want to take the tightest lower bound possible to obtain the most efficient cuts, we take : $\tilde{L}_l^i = 0$.

This enables us to formulate the McCormick envelopes:

$$\begin{cases} z_{k+1}^j z_l^i \geq \tilde{L}_l^i z_{k+1}^j \\ z_{k+1}^j z_l^i \leq U_l^i z_{k+1}^j \\ z_{k+1}^j z_l^i \geq U_l^i z_{k+1}^j + U_{k+1}^j z_l^i - U_{k+1}^j U_l^i \\ z_{k+1}^j z_l^i \leq \tilde{L}_l^i z_{k+1}^j + U_{k+1}^j z_l^i - U_{k+1}^j \tilde{L}_l^i \end{cases}$$

These inequalities yield two boundings of $A_l^i z_{k+1}^j z_l^i$, the first one being :

$$A_l^i z_{k+1}^j z_l^i \in \begin{cases} [A_l^i \tilde{L}_l^i z_{k+1}^j,\ A_l^i U_l^i z_{k+1}^j] & \text{if } A_l^i \geq 0 \\ [A_l^i U_l^i z_{k+1}^j,\ A_l^i \tilde{L}_l^i z_{k+1}^j] & \text{otherwise.} \end{cases} \tag{33}$$

And the second one being :

$$A_l^i z_{k+1}^j z_l^i \in \begin{cases} [A_l^i\big(\tilde{L}_l^i z_{k+1}^j + U_{k+1}^j z_l^i - U_{k+1}^j \tilde{L}_l^i\big),\ A_l^i\big(U_l^i z_{k+1}^j + U_{k+1}^j z_l^i - U_{k+1}^j U_l^i\big)] & \text{if } A_l^i \geq 0 \\ [A_l^i\big(U_l^i z_{k+1}^j + U_{k+1}^j z_l^i - U_{k+1}^j U_l^i\big),\ A_l^i\big(\tilde{L}_l^i z_{k+1}^j + U_{k+1}^j z_l^i - U_{k+1}^j \tilde{L}_l^i\big)] & \text{otherwise.} \end{cases} \tag{34}$$

By summing up the lower linear bounds of $A_l^i z_{k+1}^j z_l^i$ in (33) for all $l \in [0, k-1], i \in [1, n_l]$, we obtain a lower linear bound on the right term of (32) : $\sum_{l=0}^{k-1}\sum_{i=1}^{n_l} A_l^i z_{k+1}^j z_l^i + B z_{k+1}^j \leq (C_1 + B) z_{k+1}^j$. Applying the same approach for the upper bound of (33) and lower and upper bound of (34), we obtain our final set of 4 constraints:

$$\begin{cases} z_{k+1}^j(z_{k+1}^j - W_{k,u}^j z_k^u - b_k) \leq (C_1 + B) z_{k+1}^j & 28 \\ z_{k+1}^j(z_{k+1}^j - W_{k,u}^j z_k^u - b_k) \geq (C_2 + B) z_{k+1}^j & 29 \\ z_{k+1}^j(z_{k+1}^j - W_{k,u}^j z_k^u - b_k) \leq \sum_{l=0}^{k-1} \sum_{i=1}^{n_l} C_{3,l}^i z_l^i + C_{3,k+1} z_{k+1}^j + C_3 & 30 \\ z_{k+1}^j(z_{k+1}^j - W_{k,u}^j z_k^u - b_k) \geq \sum_{l=0}^{k-1} \sum_{i=1}^{n_l} C_{4,l}^i z_l^i + C_{4,k+1} z_{k+1}^j + C_4 & 31 \end{cases}$$

where

$$\begin{cases} C_1 = \sum_{l=0}^{k-1} \sum_{i=1, A_l^i \geq 0}^{n_l} A_l^i U_l^i + \sum_{l=0}^{k-1} \sum_{i=1, A_l^i < 0}^{n_l} A_l^i \tilde{L}_l^i \\ C_2 = \sum_{l=0}^{k-1} \sum_{i=1, A_l^i < 0}^{n_l} A_l^i U_l^i + \sum_{l=0}^{k-1} \sum_{i=1, A_l^i \geq 0}^{n_l} A_l^i \tilde{L}_l^i \\ C_3 = B - \sum_{l=0}^{k-1} \sum_{i=1, A_l^i \geq 0}^{n_l} A_l^i U_{k+1}^j U_l^i - \sum_{l=0}^{k-1} \sum_{i=1, A_l^i < 0}^{n_l} A_l^i \tilde{L}_l^i U_{k+1}^j \\ C_4 = B - \sum_{l=0}^{k-1} \sum_{i=1, A_l^i \geq 0}^{n_l} A_l^i U_{k+1}^j \tilde{L}_l^i - \sum_{l=0}^{k-1} \sum_{i=1, A_l^i < 0}^{n_l} A_l^i U_l^i U_{k+1}^j \\ C_{3,k+1} = \sum_{l=0}^{k-1} \sum_{i=1, A_l^i \geq 0}^{n_l} A_l^i U_l^i + \sum_{l=0}^{k-1} \sum_{i=1, A_l^i < 0}^{n_l} A_l^i \tilde{L}_l^i \\ C_{4,k+1} = \sum_{l=0}^{k-1} \sum_{i=1, A_l^i \geq 0}^{n_l} A_l^i \tilde{L}_l^i + \sum_{l=0}^{k-1} \sum_{i=1, A_l^i < 0}^{n_l} A_l^i U_l^i \\ C_{3,l}^i = C_{4,l}^i = A_l^i U_{k+1}^j \qquad \forall i \in [1, n_l], l \in [0, k-1] \end{cases}$$

Note that a wider set of constraints could be explored by varying combinations of bounds from (33) and (34). This ablation multiply by 4 the number of ReLU constraints dedicated to unstable neurons. However, it eliminates several categories of constraints related to stable active neurons, including the ReLU constraint (11), the bounding constraints (13), the triangular constraints (14), the McCormick constraints (24), and the RLT constraints (16). Notably, this leads to a quadratic reduction with respect to $n_k^a : 4\, n_k^a\, |\bar{\mathcal{J}}_K|$ constraints removed from 24, and up to $n_k^a(n_k^u + n_k^a)$ constraints removed from (16) on layer k. In sufficiently big neural networks, this quadratic reduction counterbalances the linear increase in constraints with respect to n_k^u.

B.3 Triangular constraint

We say that a neuron j of layer k of lower bound L_k^j and upper bound U_k^j is *stable active* if $L_k^j \geq 0$ and *stable inactive* if $U_k^j \leq 0$. Otherwise, a neuron is *unstable* (i.e. if $L_k^j < 0 < U_k^j$).

In order to tighten the upper bound on the ReLU activation function, a well-known constraint is the triangular constraint, which provides a convex embedding of the ReLU output. Depending on the activation status of the considered neuron j of layer k, we decompose the linear upper bound in the following set $\mathcal{T}$ of triangular inequalities:

$$z_k^j \in \mathcal{T} \Leftrightarrow \begin{cases} z_k^j \leq 0 & \text{if } j \text{ is } \textit{inactive} \\ z_k^j \leq W_k^j z_{k-1} + b_k^j & \text{if} j \text{ is } \textit{active} \\ z_k^j \leq \dfrac{U_k^j}{U_k^j - L_k^j}(W_k^j z_{k-1} + b_k^j) & \\ \quad + \dfrac{U_k^j}{U_k^j - L_k^j}(b_k^j - L_k^j) & \text{if } j \text{ is } \textit{unstable} \end{cases}$$

Combined with constraints $z_k^j \geq 0$, $z_k^j \leq \hat{z}_k^j$, where $\hat{z}_k^j$ denotes preactivation vector of neuron j of layer k, this constraint yields the exact output $z_k^j = 0$ when neuron j is inactive, and $z_k^j = W_k^j z_{k-1} + b_k^j$ when it is active.

When neuron j is unstable, the upper-bound is plotted in red in Figure 3. The equation of the red line

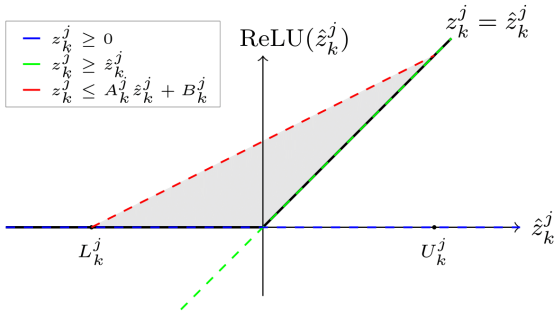

Figure 3: Triangular constraint on neuron j of layer k, where $\hat{z}_k^j = W_k^j z_{k-1} + b_k^j$ is the pre-activation vector

clearly depends on the lower and upper bounds L_k^j and U_k^j of the preactivation vector $W_k^j z_{k-1} + b_k^j$.

The triangular constraint has been shown to be limited, as it represents only the convex hull of the output of a single ReLU neuron Salman et al. (2019). Recent works have porposed convex relaxations to capture the joint behavior of multiple ReLUs. Such ideas could be explored to compute more efficient cuts in SDP models.

B.4 Coherence constraint Eq. (15)

In all our SDP models, we use the constraint between two consecutive matrices $P_k[(1 \;\; z_{k+1})(1 \;\; z_{k+1})^\top] = P_{k+1}[(1 \;\; z_{k+1})(1 \;\; z_{k+1})^\top]$ (15) relaxed as in Batten et al. (2021) and in Lan et al. (2022). For a layer k, including all coherence constraints in the model would introduce $\frac{n_k(n_k+3)}{2}$ constraints. In the relaxation of constraint (15), only the n_k linear constraints remain: $P_k[z_{k+1}] = P_{k+1}[z_{k+1}]$, preventing the number of constraints from exploding.

B.5 RLT constraint

We present here the RLT cuts (16) selected in Lan et al. (2022). These cuts contribute to tightening the relaxation, and are given below:

$$\text{Tightening cuts} \begin{cases} z_k z_{k+1} \leq L_k z_{k+1} - U_{k+1} z_k + L_k U_{k+1} & (35) \\ z_k z_{k+1} \leq L_k z_{k+1} - U_{k+1} z_k + L_k U_{k+1} & (36) \\ z_k z_{k+1} \leq L_k z_{k+1} - L_{k+1} z_k + L_k L_{k+1} & (37) \\ P_k[z_{k+1} z_{k+1}] \leq U_{k+1} z_{k+1} & (38) \\ P_k[z_{k+1} z_{k+1}^T] \leq A_k P_k[z_k z_{k+1}] + B_k P_k[z_{k+1}] & (39) \end{cases}$$

The number of these constraints is large. For a given layer k, constraints (35)–(37) scale quadratically with $n_{k+1} \times n_k$, while constraints (38) and (39) scale with $n_{k+1} \times n_{k+1}$. Including all these constraints in the SDP model would significantly increase the computing time. A heuristic is therefore needed to select only a subset of these cuts. As (38) and (39) capture *intra* layer dependencies, a heuristic selecting a subset of them is difficult to design. In contrast, since (35)–(37) represents *inter* layer dependencies, a heuristic based on the linear layer weights linking them is possible.

Only a subset of constraints (35)–(37) is finally selected, based on a given percentage p. Specifically, for each neuron j on layer $k+1$ we select $\lfloor p\, n_k \rfloor$ cuts. The heuristic sorts the absolute value of the weights $|W^j_{k+1}|$, and selects the neurons corresponding to the top $\lfloor p\, n_k \rfloor$ entries in the sorted vector. This selection is based on the full size of layer k regardless of whether neurons have been pruned. More precisely, in the context of a full ablation, we do not select $\lfloor p\, n^u_k \rfloor$ RLT cuts but $\lfloor p\, n_k \rfloor$ ones.

B.6 Tightening cuts for SDP_u

For simplicity and clarity, we used the full logits z^j_K in constraints (26) (27) (24). Note that these logits are not variables of our model. To obtain the full constraints in our model, we need to substitute each logit by its linear expression with respect to the penultimate layer variables : $z^j_K = W^j_K z_{K-1} + b^j_K$.

Furthermore, for the sake of clarity in constraints (23) (24) (26) (27 (25), we omit explicit matrix indexation. To recover the full constraint, one must, for example, replace variables such as $\beta_i\beta_j$ by $P_{K-2}[\beta_i\beta_j]$. Combining this with the logit expression, z^j_K should be replaced by $W^j_K P_{K-2}[z_{K-1}] + b^j_K$, and $\beta_j z^j_K$ by $W^j_K P_{K-2}[\beta_j z_{K-1}] + b^j_K P_{K-2}[\beta_j]$.

C IMPLEMENTATION DETAILS

All networks have been trained with a batch size of 128, the Adam optimizer, and a learning rate of 0.001. For reproducibility, we show the details of the adversarial training in table 3. We used the optimizer Adam, a learning rate $lr = 0.01$, and batch sizes of 128. All PGD attacks were used with number of steps $= 40$, a random start. We denote by `6x100-5`, `6x100-20`, `6x100-50`, `6x100-67` networks used in experiment 3.

Network	**Architecture**	**Adversarial Training**		**Accuracy**
		Epochs	Adversarial attack	
`6x100`	784-6x100-10	200	PGD ($\epsilon = 0.3$, $\alpha = 0.01$)	95.5
`6x200`	784-6x200-10	200	PGD ($\epsilon = 0.3$, $\alpha = 0.01$)	96.1
`9x100`	784-9x100-10	200	PGD ($\epsilon = 0.3$, $\alpha = 0.01$)	95.2
`9x200`	784-9x200-10	200	PGD ($\epsilon = 0.3$, $\alpha = 0.01$)	96.9
`6x100-5`	784-6x100-5	100	PGD ($\epsilon = 0.3$, $\alpha = 0.01$)	97.6
`6x100-20`	784-6x100-20	100	PGD ($\epsilon = 0.3$, $\alpha = 0.01$)	86.0
`6x100-50`	784-6x100-50	100	PGD ($\epsilon = 0.3$, $\alpha = 0.01$)	76.7
`6x100-67`	784-6x100-67	100	PGD ($\epsilon = 0.3$, $\alpha = 0.01$)	75.2

Table 3: Networks used in our three experiments.

Network	ϵ	**Stability study**				Running targets
		Stable		Unstable	Total	
		Inactives	Actives			
`6x100`	0.026	**283**	**224**	**93**	**600**	4.0
		47%	37%	16%	100%	
`6x200`	0.015	**713**	**287**	**200**	**1200**	6.8
		59%	24%	17%	100%	
`9x100`	0.026	**364**	**225**	**312**	**900**	7.1
		40%	25%	35%	100%	
`9x200`	0.015	**1024**	**312**	**464**	**1800**	7.6
		57%	17%	26%	100%	
`6x100-5`	0.05	**261**	**119**	**220**	**600**	3.25
		44%	20%	36%	100%	
`6x100-20`	0.05	**243**	**157**	**201**	**600**	14.44
		40%	26%	33%	100%	
`6x100-50`	0.05	**188**	**117**	**295**	**600**	48.8
		31%	20%	49%	100%	
`6x100-67`	0.05	**203**	**127**	**271**	**600**	60.1
		34%	21%	45%	100%	

Table 4: Mean number of stable active, stable inactive and unstable neurons computed on the data of our experiments (100 data for `6x100-6x200-9x100-9x200`, 5 data for `6x100-5`, 20 data for `6x100-20`, 50 data for `6x100-50`, and 67 data for `6x100-67`).

D ADDITIONNAL STATE OF THE ART

D.1 OTHER INCOMPLETE VERIFICATION USING SDP

Other related efforts include the development of dual solvers tailored to the SDP formulation Dathathri et al. (2018). Other works have created a Branch and Bound framework based on a rewriting of the matrix coefficient of the quadratic constraint of the ReLU $BB - SDPt$ **?**. While these approaches differ significantly from ours—making them difficult to reproduce—we observe that they primarily improve certification rates rather than scalability. In **?**, we can estimate the computing method time multiplying the certification time of the model (targeted) by an underestimation of the number of running targets. We see that our model is faster for all common networks structure in our experiments (`6x100`, `6x200`, `9x100`).

Some recent works have adressed different settings. Notably, some have incorporated SDP-based bounds to into the Branch and Bound framework commonly used with β-CROWN. They observed that bounds computed with β-CROWN are bad with the norm 2 Chiu et al. (2025).

D.2 SDP RELAXATIONS

The two main criterion to design a relaxation is its computation time and its tightness. Indeed, the tighter the relaxation, the better the lower bound. Existing relaxation techniques for quadratic problems are mainly based on linearization or on semi-definite programming. To compute a linear relaxation, the quadratic functions are reformulated as convex equivalent functions in an extended space of variables. More precisely, new variables Z_{ij} are introduced for all (i, j), that are meant to satisfy the equalities $Z_{ij} = z_i z_j$. The linearization is then obtained by relaxation of the later non-convex equalities, for instance by linear constraints (see for instance McCormick (1976); Sherali & Adams (2013); Yajima & Fujie (1998)). Using semi-definite relaxations for quadratic programming was also widely studied Anstreicher (2009); Chen & Burer (2012); Burer & Vandenbussche (2008; 2009); Vandenbussche & Nemhauser (2005a;b). A semi-definite relaxation of a quadratic optimization problem can be obtained by lifting z to a symmetric matrix $Z = zz^\top$ where the later non-convex constraints are relaxed to $Z - zz^\top \succeq 0$. Note that, since in a DNN only layers k and $k + 1$ are linked by Relu Constraints, a chordal decomposition of the variable matrix Z into $|\mathcal{K}|$ block diagonal matrices is possible Batten et al. (2021). This standard semi-definite relaxation is often referred to as "Shor's" relaxation. In Anstreicher (2009), the "Shor's plus RLT" relaxation was introduced, where the convex envelopes of the quadratic terms McCormick (1976) where added to the later relaxation. We detailed the RLT cuts used in Lan et al. (2022) in section B.5.

D.3 COMPLETE VERIFICATION

Ideal verification is complete, ensuring that all answers are reliable. However, due to the complexity of the problem, fully achieving such verification is often constrained. Works using Satisfiability Modulo Theory Ehlers (2017) like Reluplex Katz et al. (2017) or Marabou Katz et al. (2019) Wu et al. (2024) have been developed, but are not currently scalable. Nevertheless, they are very precise and give formal proof of robustness or useful counterexamples when working on a sufficiently small network. Some works have introduced Mixed Integer Programming formulations (MIP), see Fischetti & Jo (2018) Cheng et al. (2017), but the direct resolution of these models without relaxations are also not scalable. Most efficient complete verification rely on Brand&Bound Bunel et al. (2020)- Ferrari et al. (2022)- Jaeckle et al. (2021)- Lu & Kumar (2019), whose relaxation of the certification problem is fast heuristics like CROWN. They have been improved by smart branching: splitting on the activation or not activation of a set of neurons has been exponentially faster than splitting on the input ball. Some methods have improve¡d ReLU splitting to better choose neurons for branching decision Henriksen & Lomuscio (2021). When reaching a certain depth of the tree, a relatively fast MIP is solved (with few binary variables as the activation of most neurons is fixed), to prune a branch without exploring all its content. It also helps to avoid impossible activation patterns in practice, which may not be seen by heuristic methods like bound propagation, guaranteeing the soundness of the algorithm. The resolution of these MILP has been further improved with cutting planes Zhang et al. (2022).

D.4 ADVERSARIAL TRAINING

In this section, we present the adversarial training used in order to create robust networks. Madry introduced adversarial training Madry et al. (2019) by adding a maximisation problem into the common training minimisation problem:

$$\min_{\forall(x,y)\in\mathcal{X}} \max_{z\in\mathcal{B}_\epsilon(x)} \mathcal{L}(z,y)$$

where the inner maximisation represents the computation of the worst adversarial attack. It is approached by heuristics, most of them are based on gradient descents, Projected Gradient Descent (PGD) Madry et al. (2019), or its variants (FGSM Goodfellow et al. (2015), IGS Kurakin et al. (2017)), JSMA Papernot et al. (2015) for the norm $\|\|\|_0$, DeepFool Moosavi-Dezfooli et al. (2016) for $\|\|\|_2$) and have been improved since Carlini & Wagner (2017). Specifically, we train our model using the most classic adversarial attack: PGD, and compute untargeted adversarial attacks with it.