A Proofs for Section 3.2

We present rigorous proofs for Lemma 3.2, Theorems 3.3 and 3.4 in Section 3.2, justifying the *soundness* and *optimality* of our VERIX approach. For better readability, we repeat each lemma and theorem before their corresponding proofs.

A.1 Proof for Lemma 3.2

Lemma 3.2. If the CHECK sub-procedure is *sound*, then, at the end of each `for-loop` iteration (Lines 7–12) in Algorithm 1, the *irrelevant* set of indices $\mathbf{B}$ satisfies

$$(\left\|\hat{\chi}^{\mathbf{B}} - \chi^{\mathbf{B}}\right\|_p \leq \epsilon) \wedge (\hat{\chi}^{\Theta\setminus\mathbf{B}} = \chi^{\Theta\setminus\mathbf{B}}) \Rightarrow |\hat{c} - c| \leq \delta. \tag{6}$$

Proof. Recall that the sub-procedure CHECK is *sound* means the deployed automated reasoner returns `True` only if the specification actually holds. That is, from Line 10 we have

$$\phi \Rightarrow |\hat{c} - c| \leq \delta$$

holds on network f. Simultaneously, from Lines 8 and 9 we know that, to check the current feature χ^i of the traversing order π, the pre-condition ϕ contains

$$\phi \mapsto (\left\|\hat{\chi}^{\mathbf{B}^+} - \chi^{\mathbf{B}^+}\right\|_p \leq \epsilon) \wedge (\hat{\chi}^{\Theta\setminus\mathbf{B}^+} = \chi^{\Theta\setminus\mathbf{B}^+}).$$

Specifically, we prove this through induction on the number of iteration i. When i is 0, pre-condition ϕ is initialized as $\top$ and the specification holds trivially. In the inductive case, suppose CHECK returns `False`, then the set $\mathbf{B}$ is unchanged as in Line 12. Otherwise, if CHECK returns `True`, which makes `HOLD` become `True`, then the current feature index i is added into the irrelevant set of feature indices $\mathbf{B}$ as in Line 11, with such satisfying specification

$$(\left\|\hat{\chi}^{\mathbf{B}^+} - \chi^{\mathbf{B}^+}\right\|_p \leq \epsilon) \wedge (\hat{\chi}^{\Theta\setminus\mathbf{B}^+} = \chi^{\Theta\setminus\mathbf{B}^+}) \Rightarrow |\hat{c} - c| \leq \delta.$$

As the iteration proceeds, each time CHECK returns `True`, the irrelevant set $\mathbf{B}$ is augmented with the current feature index i, and the specification always holds as it is explicitly checked by the CHECK reasoner. □

A.2 Proof for Theorem 3.3

Theorem 3.3 (Soundness). If the CHECK sub-procedure is *sound*, then the value $\mathbf{x}^{\mathbf{A}}$ returned by Algorithm 1 is a *robust* explanation – this satisfies Equation (1) of Definition 2.1.

Proof. The `for-loop` from Line 6 indicates that Algorithm 1 goes through every each feature $\mathbf{x}^i$ in input $\mathbf{x}$ by traversing the set of indices $\Theta(\mathbf{x})$. Line 5 means that π is one such instance of ordered traversal. When the iteration ends, all the indices in $\Theta(\mathbf{x})$ are either put into the irrelevant set of indices by $\mathbf{B} \mapsto \mathbf{B}^+$ as in Line 11 or the explanation index set by $\mathbf{A} \mapsto \mathbf{A} \cup \{i\}$ as in Line 12. That is, $\mathbf{A}$ and $\mathbf{B}$ are two disjoint index sets forming $\Theta(\mathbf{x})$; in other words, $\mathbf{B} = \Theta(\mathbf{x}) \setminus \mathbf{A}$. Therefore, combined with Lemma 3.2, when the reasoner CHECK is *sound*, once iteration finishes we have the following specification

$$(\left\|\hat{\chi}^{\mathbf{B}} - \chi^{\mathbf{B}}\right\|_p \leq \epsilon) \wedge (\hat{\chi}^{\Theta\setminus\mathbf{B}} = \chi^{\Theta\setminus\mathbf{B}}) \Rightarrow |\hat{c} - c| \leq \delta. \tag{7}$$

holds on network f, where $\hat{\chi}^{\mathbf{B}}$ is the variable representing all the possible assignments of irrelevant features $\mathbf{x}^{\mathbf{B}}$, i.e., $\forall\, \mathbf{x}^{\mathbf{B}'}$, and the pre-condition $\hat{\chi}^{\Theta\setminus\mathbf{B}} = \chi^{\Theta\setminus\mathbf{B}}$ fixes the values of the explanation features of an instantiated input $\mathbf{x}$. Meanwhile, the post-condition $|\hat{c} - c| \leq \delta$ where $c \mapsto f(\mathbf{x})$ as in Line 3 ensures prediction invariance such that δ is 0 for classification and otherwise a pre-defined allowable amount of perturbation for regression. To this end, for some specific input $\mathbf{x}$ we have the following property

$$\forall\, \mathbf{x}^{\mathbf{B}'}.\ (\left\|\mathbf{x}^{\mathbf{B}'} - \mathbf{x}^{\mathbf{B}}\right\|_p \leq \epsilon) \Rightarrow |f(\mathbf{x}') - f(\mathbf{x})| \leq \delta. \tag{8}$$

holds. Here we prove by construction. According to Equation (1) of Definition 2.1, if the irrelevant features $\mathbf{x}^{\mathbf{B}}$ satisfy the above property, then we call the rest features $\mathbf{x}^{\mathbf{A}}$ a *robust* explanation with respect to network f and input $\mathbf{x}$. □

A.3 Proof for Theorem 3.4

Theorem 3.4 (Optimality). If the CHECK sub-procedure is *sound* and *complete*, then the *robust* explanation $\mathbf{x^A}$ returned by Algorithm 1 is *optimal* – this satisfies Equation (2) of Definition 2.1.

Proof. We prove this by contradiction. From Equation (2) of Definition 2.1, we know that explanation $\mathbf{x^A}$ is optimal if, for any feature χ in the explanation, there always exists an ϵ-perturbation on χ and the irrelevant features $\mathbf{x^B}$ such that the prediction alters. Let us suppose $\mathbf{x^A}$ is not optimal, then there exists a feature χ in $\mathbf{x^A}$ such that no matter how to manipulate this feature χ into χ' and the irrelevant features $\mathbf{x^B}$ into $\mathbf{x^{B'}}$, the prediction always remains the same. That is,

$$\exists\, \chi \in \mathbf{x^A}.\ \forall\, \mathbf{x^{B'}}, \chi'.\ \left\|(\mathbf{x^B} \oplus \chi) - (\mathbf{x^{B'}} \oplus \chi')\right\|_p \leq \epsilon \Rightarrow |f(\mathbf{x}) - f(\mathbf{x'})| \leq \delta, \tag{9}$$

where $\oplus$ denotes concatenation of two features. When we pass this input $\mathbf{x}$ and network f into the VERIX framework, suppose Algorithm 1 examines this feature χ at the i-th iteration, then as in Line 7, the current irrelevant set of indices is $\mathbf{B}^+ \mapsto \mathbf{B} \cup \{i\}$, and accordingly the pre-conditions are

$$\phi \mapsto (\left\|\hat{\chi}^{\mathbf{B}\cup\{i\}} - \chi^{\mathbf{B}\cup\{i\}}\right\|_p \leq \epsilon) \wedge (\hat{\chi}^{\Theta\setminus(\mathbf{B}\cup\{i\})} = \chi^{\Theta\setminus(\mathbf{B}\cup\{i\})}). \tag{10}$$

Because $\hat{\chi}^{\mathbf{B}\cup\{i\}}$ is the variable representing all the possible assignments of irrelevant features $\mathbf{x^B}$ and the i-th feature χ, i.e., $\forall\, \mathbf{x^{B'}}, \chi'$, and meanwhile

$$\hat{\chi}^{\Theta\setminus(\mathbf{B}\cup\{i\})} = \chi^{\Theta\setminus(\mathbf{B}\cup\{i\})} \tag{11}$$

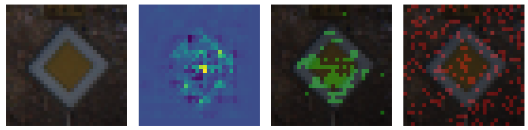
indicates that the other features are fixed with specific values of this $\mathbf{x}$. Thus, with $c \mapsto f(\mathbf{x})$ in Line 3, we have the specification $\phi \Rightarrow |\hat{c} - c| \leq \delta$ holds on input $\mathbf{x}$ and network f. Therefore, if the reasoner CHECK is *sound* and *complete*,

$$\text{CHECK}(f, \phi \Rightarrow |\hat{c} - c| \leq \delta) \tag{12}$$

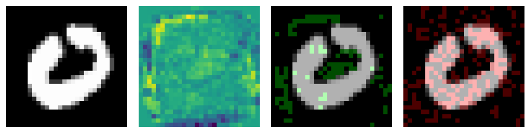
will always return `True`. Line 10 assigns `True` to `HOLD`, and index i is then put into the irrelevant set $\mathbf{B}$ thus i-th feature χ in the irrelevant features $\mathbf{x^B}$. However, based on the assumption, feature χ is in explanation $\mathbf{x^A}$, so χ is in $\mathbf{x^A}$ and $\mathbf{x^B}$ simultaneously – a contradiction occurs. Therefore, Theorem 3.4 holds. □

B Supplementary experimental results

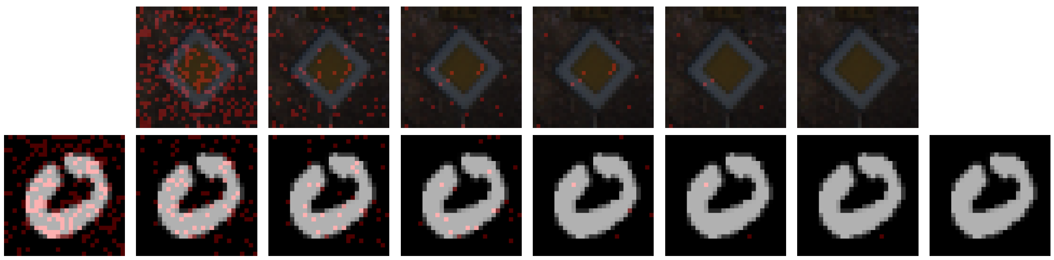
B.1 Sensitivity vs. random traversal to generate explanations



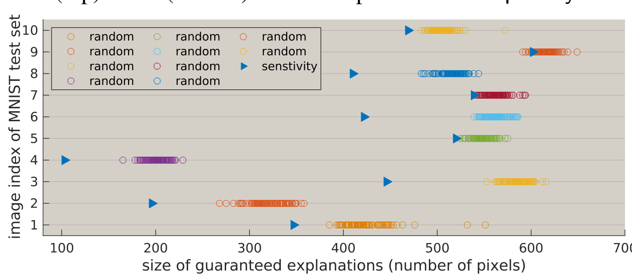
(a) GTSRB “priority road”; sensitivity; explanations from sensitivity (green) and random (red) traversals.



(b) MNIST “0”; sensitivity; explanations from sensitivity (green) and random (red) traversals.



(c) Empty intersection of 6 (top) and 8 (bottom) random explanations for “priority road” and “0”, respectively.



(d) Sensitivity vs. random traversals in explanation size. Each blue triangle denotes 1 deterministic explanation from sensitivity ranking, and each bunch of circles represents 100 explanations from random traversals.

Figure 10: VERIX explanations when using *sensitivity* (green) and random (red) traversals.

To show the advantage of the *sensitivity* traversal, Figure 10 compares VERIX explanations using sensitivity-based and random traversal orders. The first column of Figures 10a and 10b shows the original image; the second a heatmap of the sensitivity (with deletion $\mathcal{T}(\chi) = 0$ for GTSRB and reversal $\mathcal{T}(\chi) = \overline{\chi} - \chi$ for MNIST because deleting background pixels of MNIST images may contribute to little confidence change as they often have zero values); and the third and fourth columns show explanations using the sensitivity and random traversals, respectively. Sensitivity, as shown in the heatmaps, prioritizes pixels that have more influence on the network’s prediction. In contrast, a random ranking is simply a shuffling of all the pixels. We observe that the sensitivity traversal generates smaller and more sensible explanations. Furthermore, we also explore the idea of using intersections of explanations generated from random traversals. Specifically, for both images, we randomly traverse all input features 10 times and produce 10 explanations. In Figure 10c, we show the result of the first random explanation, followed by the result of intersecting this explanation with more and more random explanations. The end result is an empty set (last one in each row). This strongly emphasizes the necessity of a sensible traversal, for which we propose the feature-level sensitivity traversal. In Figure 10d, we compare explanation sizes for the first 10 images (to avoid potential selection bias) of the MNIST test set. For each image, we show 100 random traversal explanations compared to the deterministic explanation from sensitivity traversal. We observe that the latter is almost always smaller, often significantly so, suggesting that sensitivity-based traversals are a reasonable heuristic for attempting to approach globally optimal explanations.

B.2 Runtime performance

Table 3: Average execution time (seconds) of CHECK and VeriX for *complete* verification. In particular, magnitude ϵ is set to 3% across the `Dense`, `Dense (large)`, `CNN` models and the MNIST, TaxiNet, GTSRB datasets for sensible comparison.

	`Dense`		`Dense (large)`		`CNN`	
	CHECK	VeriX	CHECK	VeriX	CHECK	VeriX
MNIST (28×28)	0.013	160.59	0.055	615.85	0.484	4956.91
TaxiNet (27×54)	0.020	114.69	0.085	386.62	2.609	8814.85
GTSRB ($32 \times 32 \times 3$)	0.091	675.04	0.257	1829.91	1.574	12935.27

Table 4: Average execution time (seconds) of CHECK and VeriX for *incomplete* verification. Magnitude ϵ is 3% for both MNIST-`sota` and GTSRB-`sota` models.

	# ReLU	# MaxPool	CHECK	VeriX
MNIST-`sota`	50960	5632	2.31	1841.25
GTSRB-`sota`	106416	5632	8.54	8770.15

Figure 11: Sound but *incomplete* CHECK procedure CROWN contributes to robust but *not optimal* (larger than necessary) VeriX explanations for the convolutional network MNIST-`sota`.

We analyze the empirical time *complexity* of our VeriX approach in Table 3. The model structures are described in Appendix D.4. Typically, the individual pixel checks (CHECK) return a definitive answer (`True` or `False`) within a second on dense models and in a few seconds on convolutional networks. For image benchmarks such as MNIST and GTSRB, larger inputs or more complicated models result in longer (pixel- and image-level) execution times for generating explanations. As for TaxiNet as a regression task, while its pixel-level check takes longer than that of MNIST, it is actually faster in total time on dense models because TaxiNet does not need to check against other labels.

The *scalability* of VeriX can be improved if we perform incomplete verification, for which we re-emphasize that the soundness of the resulting explanations is not undermined though optimality is no longer guaranteed, i.e., they may be larger than necessary. To illustrate, we deploy the incomplete CROWN [64] analysis (implemented in Marabou) to perform the CHECK sub-procedure. Table 4 reports the runtime performance of VeriX when using incomplete verification on state-of-the-art network architectures with hundreds of thousands of neurons. See model structures in Appendix D, Tables 6 and 8. Moreover, in Figure 11, we include some example explanations for the convolutional model MNIST-`sota` when using the sound but incomplete CHECK procedure CROWN. We can see that they indeed appear larger than the optimal explanations when the complete Marabou reasoner is used. We remark that, as soundness of the explanations is not undermined, they still provide guarantees against perturbations on the irrelevant pixels. Interestingly, MNIST explanations on convolutional models tend to be less scattered than these on fully-connected models, as shown in Figures 4b and 5b, due to the effect of convolutions. In general, the scalability of VeriX will grow with that of verification tools, which has improved significantly in the past several years as demonstrated by the results from the Verification of Neural Networks Competitions (VNN-COMP) [3].

C Supplementary related work

C.1 Related work (cont.)

Continued from Section 5, our work expands on [31] in four important ways: (i) we focus on ϵ-ball perturbations and perception models, whose characteristics and challenges are different from those of NLP models; (ii) whereas [31] simply points to existing work on hitting sets and minimum satisfying assignments for computing OREs, we provide a detailed algorithm with several illustrative examples, and include a concrete traversal heuristic that performs well in practice; we believe these details are useful for anyone wanting to produce a working implementation; (iii) we note for the first time the relationship between OREs and counterfactual explanations; and (iv) we provide an extensive evaluation on a variety of perception models. We also note that in some aspects, our work is more limited: in particular, we use a simpler definition of ORE (without a cost function) as our algorithm is specialized for the case of finding explanations with the fewest features.

We discuss some further related work in the formal verification community that are somewhat centered around interpretability or computing minimal explanations. [13] uses formal techniques to identify input regions around an adversarial example such that all points in those regions are also guaranteed to be adversarial. Their work improves upon previous work on identifying *empirically robust* adversarial regions, where points in the regions are empirically likely to be adversarial. Analogously, our work improves upon informal explanation techniques like Anchors. [11] is similar to [13] in that it also computes pre-images of neural networks that lead to bad outputs. In contrast, we compute a subset of input features that preserves the neural network output. [65] is more akin to our work, with subtle yet important differences. Their goal is to identify a *minimal* subset of input features that when *corrected*, *changes* a network's prediction. In contrast, our goal is to find a *minimal* subset of input features that when *fixed*, *preserves* a network's prediction. These two goals are related but not equivalent: given a correction set found by [65], a sound but non-minimal VeriX explanation can be obtained by fixing all features not in the correction set along with one of the features in the correction set. Symmetrically, given a VeriX explanation, a sound but non-minimal correction can be obtained by perturbing all the features not in the explanation along with one of the features in the explanation. This relation is analogous to that between minimal correction sets and minimal unsatisfiable cores in constraint satisfaction (e.g., [33]). Both are considered standard explanation strategies in that field.

C.2 Verification of neural networks

Researchers have investigated how automated reasoning can aid verification of neural networks with respect to formally specified properties [34, 20], by utilizing reasoners based on abstraction [64, 44, 16, 47, 39, 52, 53, 2, 63, 55, 57] and search [14, 28, 29, 21, 54, 46, 19, 7, 12, 42, 60, 59, 5, 30, 15, 56, 58]. Those approaches mainly focus on verifying whether a network satisfies a certain pre-defined property (e.g., robustness), i.e., either prove the property holds or disprove it with a counterexample. However, this does not shed light on *why* a network makes a specific prediction. In this paper, we take a step further, repurposing those verification engines as sub-routines to inspect the decision-making process of a model, thereby explaining its behavior (through the presence or absence of certain input features). The hope is that these explanations can help humans better interpret machine learning models and thus facilitate appropriate deployment.

D Model specifications

Apart from those experimental settings in Section 4, we include detailed model specifications for reproducibility and reference purposes. Although evaluated on the MNIST [32], GTSRB [45], and TaxiNet [27] image datasets – MNIST and GTSRB in classification and TaxiNet in regression, our VeriX framework can be generalized to other machine learning applications such as natural language processing. As for the sub-procedure CHECK of Algorithm 1, while VeriX can potentially incorporate existing automated reasoners, we deploy the neural network verification tool Marabou [29]. While it supports various model formats such as .pb from TensorFlow [1] and .h5 from Keras [8], we employ the cross platform .onnx format for better Python API support. When importing a model with softmax as the final activation function, we remark that, for the problem to be *decidable*, one needs to specify the `outputName` parameter of the `read_onnx` function as the pre-softmax logits. As a workaround for this, one can also train the model without softmax in the last layer and instead use the SoftmaxLoss loss function from the tensorflow_ranking package. Either way, VeriX produces consistent results.

D.1 MNIST

For MNIST, we train a fully-connected feed-forward neural network with 3 dense layers activated with ReLU (first 2 layers) and softmax (last classification layer) functions as in Table 5, achieving 92.26% accuracy. While the MNIST dataset can easily be trained with accuracy as high as 99.99%, we are more interested in whether a very simple model as such can extract sensible explanations – the answer is yes. Meanwhile, we also train several more complicated MNIST models, and observe that their optimal explanations share a common phenomenon such that they are relatively more scattered around the background compared to the other datasets. This cross-model observation indicates that MNIST models need to check both the presence and absence of white pixels to recognize the handwritten digits correctly. Besides, to show the scalability of VeriX, we also deploy incomplete verification on state-of-the-art model structure as in Table 6.

D.2 GTSRB

As for the GTSRB dataset, since it is not as identically distributed as MNIST, to avoid potential distribution shift, instead of training a model out of the original 43 categories, we focus on the top first 10 categories with highest occurrence in the training set. This allows us to obtain an appropriate model with high accuracy – the convolutional model we train as in Table 7 achieves a test accuracy of 93.83%. It is worth mentioning that, our convolutional model is much more complicated than the simple dense model in [24], which only contains one hidden layer of 15 or 20 neurons trained to distinguish two MNIST digits. Also, as shown in Table 4 of Section B.2, we report results on the state-of-the-art GTSRB classifier in Table 8.

D.3 TaxiNet

Apart from the classification tasks performed on those standard image recognition benchmarks, our VeriX approach can also tackle regression models, applicable to real-world safety-critical domains. In this vision-based autonomous aircraft taxiing scenario [27] of Figure 9, we train the regression model in Table 9 to produce an estimate of the cross-track distance (in meters) from the ownship to the taxiway centerline. The TaxiNet model has a mean absolute error of 0.824 on the test set, with no activation function in the last output layer.

D.4 Dense, Dense (large), and CNN

In Section B.2, we analyze execution time of VeriX on three models with increasing complexity: `Dense`, `Dense (large)`, and `CNN` as in Tables 10, 11, and 12, respectively. To enable a fair and sensible comparison, those three models are used across the MNIST, TaxiNet, and GTSRB datasets with only necessary adjustments to accommodate each task. For example, in all three models $\mathtt{h} \times \mathtt{w} \times \mathtt{c}$ denotes different input size $\mathtt{height} \times \mathtt{width} \times \mathtt{channel}$ for each dataset. For the activation function of the last layer, softmax is used for MNIST and GTSRB while TaxiNet as a regression task needs no such activation. Finally, TaxiNet deploys he_uniform as the `kernel_initializer` parameter in the intermediate dense and convolutional layers for task specific reason.

Table 5: Structure for the MNIST classifier.

Layer Type	Parameter	Activation
Input	$28 \times 28 \times 1$	–
Flatten	–	–
Fully Connected	10	ReLU
Fully Connected	10	ReLU
Fully Connected	10	softmax

Table 7: Structure for the GTSRB classifier.

Type	Parameter	Activation
Input	$32 \times 32 \times 3$	–
Convolution	$3 \times 3 \times 4$ (1)	–
Convolution	$2 \times 2 \times 4$ (2)	–
Fully Connected	20	ReLU
Fully Connected	10	softmax

Table 6: Structure for the MNIST-`sota` classifier.

Type	Parameter	Activation
Input	$28 \times 28 \times 1$	–
Convolution	$3 \times 3 \times 32$	ReLU
Convolution	$3 \times 3 \times 32$	ReLU
MaxPooling	2×2	–
Convolution	$3 \times 3 \times 64$	ReLU
Convolution	$3 \times 3 \times 64$	ReLU
MaxPooling	2×2	–
Flatten	–	–
Fully Connected	200	ReLU
Dropout	0.5	–
Fully Connected	200	ReLU
Fully Connected	10	softmax

Table 8: Structure for the GTSRB-`sota` classifier.

Type	Parameter	Activation
Input	$28 \times 28 \times 1$	–
Convolution	$3 \times 3 \times 32$	ReLU
Convolution	$3 \times 3 \times 32$	ReLU
Convolution	$3 \times 3 \times 64$	ReLU
MaxPooling	2×2	–
Convolution	$3 \times 3 \times 64$	ReLU
Convolution	$3 \times 3 \times 64$	ReLU
MaxPooling	2×2	–
Flatten	–	–
Fully Connected	200	ReLU
Dropout	0.5	–
Fully Connected	200	ReLU
Fully Connected	10	softmax

Table 9: Structure for the TaxiNet model.

Type	Parameter	Activation
Input	$27 \times 54 \times 1$	–
Flatten	–	–
Fully Connected	20	ReLU
Fully Connected	10	ReLU
Fully Connected	1	–

Table 10: Structure for the `Dense` model.

Layer Type	Parameter	Activation
Input	$\mathtt{h} \times \mathtt{w} \times \mathtt{c}$	–
Flatten	–	–
Fully Connected	10	ReLU
Fully Connected	10	ReLU
Fully Connected	10 / 1	softmax / –

Table 11: Structure for `Dense` (`large`).

Layer Type	Parameter	Activation
Input	$\mathtt{h} \times \mathtt{w} \times \mathtt{c}$	–
Flatten	–	–
Fully Connected	30	ReLU
Fully Connected	30	ReLU
Fully Connected	10 / 1	softmax / –

Table 12: Structure for the `CNN` model.

Layer Type	Parameter	Activation
Input	$\mathtt{h} \times \mathtt{w} \times \mathtt{c}$	–
Convolution	$3 \times 3 \times 4$	–
Convolution	$3 \times 3 \times 4$	–
Fully Connected	20	ReLU
Fully Connected	10 / 1	softmax / –