机器学习 周志华 chap 11、chap 12 知识点总结

目录 /CONTENTS

,

Chap 11 Part 1 特征选择

- **相关特征**：对当前学习任务有用的属性
- **无关特征**：对当前学习任务没什么用的属性
- **冗余特征**：包含的信息能从其他特征中推演出来。

  冗余特征不一定是无用的，若某个冗余特征恰好对应了完成学习任务所需的中间概念，则该冗余特征是有益的。

- **特征选择**：从给定的特征集合中选择出相关特征子集的过程，包括特征子集搜索机制和子集评价机制。
- **特征选择的原因**
  - 属性过多会造成维数灾难问题，特征选择和降维都是处理高维数据的技术；
  - 去除不相关特征往往会降低学习任务的难度。
- **关键**：特征选择过程必须确保不丢失重要特征。
- **特征选择的方法**

  从初始特征集合产生选择子集并评价它的好坏，基于评价结果产生下一个候选子集再评价，持续此过程直到无法找到更好的候选子集为止。
- **常用特征选择方法**：过滤式、包裹式和嵌入式

,

Chap 11 Part 1 特征选择

- **关键环节**
  - 子集搜索问题：如何根据评价结果获取下一个候选特征子集
  - 子集评价问题：如何评价候选子集的好坏
- **解决方法**

子集搜索问题

- 前向搜索：逐渐增加相关特征，初始时将每个特征看做一个候选集，对单特征子集进行评价，将最优特征子集作为第一轮的选定集；在上一轮的选定集中加入一个特征，并重新评价；直到本轮的最优候选子集不如上一轮的选定集，将上一轮的特征集合作为特征选择结果。
- 后向搜索：每次去掉一个无关特征
- 双向搜索：每一轮逐渐增加选定相关特征（这些特征在后续轮中确定不会被去除）、同时减少无关特征

缺点：仅考虑了使本轮最优。

子集评价问题

- 特征子集A确定了对数据集D的一个划分，样本标记信息Y对应着D的真实划分，子集评价就是估算这两个划分之间的差异，差异越小特征选择越好。
- **判断差异的指标**：信息熵、不合度量、相关系数等。

- 过滤式方法先对数据集进行特征选择，再训练学习器，特征选择与过程与后续学习器无关。

- **代表方法**

Relief

- 该方法设计了一个“相关统计量”来度量特征的重要性，该统计量是一个向量，其每个分量分别对应一个初始特征，特征子集的重要性由子集中每个特征所对应的相关统计量分量之和决定。
- 指定阈值τ，选择比τ大的相关统计量分量所对应的特征即可；也可指定欲选取的特征个数k然后选择相关统计量分量最大的k个特征。
- **关键**：如何确定相关统计量
- **应用**：二分类问题
- **优点**：Relief的时间开销随采样次数以及原始特征数线性增长，是一个运行效率很高的过滤式特征选择算法。

Relief-F

- **应用**：多分类问题。

Chap 11 Part 2 包裹式选择

- 包裹式选择直接把最终将要使用的学习器的性能作为特征子集的评价准则。
- 优点：从最终学习器性能来看，包裹式特征选择比过滤式特征选择更好。
- 缺点：由于在特征选择过程中需多次训练学习器，计算开销较大。

代表方法 LVW

- LVW是一个典型的包裹式特征选择方法，它在拉斯维加斯方法（随机化方法）框架下使用随机策略进行子集搜索，并以最最终分类器的误差作为特征子集评价标准。
- **缺点**：若有运行时间的限制，可能给不出解

Chap 11 Part 4 嵌入式选择

- 嵌入式选择是将特征选择过程与学习器训练过程融为一体，两者在同一个优化过程中完成，即在学习器训练过程中自动地进行了特征选择。

  考虑线性回归模型，以平方误差为损失函数，则优化目标为：

$$\min_{\omega} \sum_{i=1}^{m} \left(y_i - w^T x_i\right)^2$$

  当样本特征很多而样本数相对较少时，上式容易陷入过拟合。
  为缓解过拟合：可引入正则化。

  L_2正则化：

$$\min_{\omega} \sum_{i=1}^{m} \left(y_i - w^T x_i\right)^2 + \lambda \| w \|_2^2$$

  L_1正则化：

$$\min_{\omega} \sum_{i=1}^{m} \left(y_i - w^T x_i\right)^2 + \lambda \| w \|_1$$

- L_1范数正则化比L_2范数正则化更容易获得稀疏解，即求得的w会有更少的非零向量。
- L_1正则化问题的求解可使用近端梯度下降（PGD）。

- **稀疏表示**

  把数据集D考虑成一个矩阵，其每行对应于一个样本，每列对应于一个特征，特征选择所考虑的问题是特征具有“稀疏性”，去除与当前任务无关的列可以降低学习难度、计算和存储的开销，学得模型的可解释性也会提高。

- **优点**：当样本数据是一个稀疏矩阵时，对学习任务来说会有不少的好处，例如很多问题变得线性可分，储存更为高效等。这便是稀疏表示与字典学习的基本出发点。

- **字典学习**

  亦称稀疏编码，找到能将普通稠密表达的样本转化为合适的稀疏表示形式的字典，对样本进行稀疏表达。

Chap 11 Part 6 压缩感知

- 压缩感知就是基于部分信息恢复全部信息，压缩感知的前提是已知的信息具有稀疏表示。

分为“感知测量”和“恢复重构”两个阶段：

- 感知测量关注如何对原始信号进行处理以获得稀疏样本表示，涉及傅里叶变换、小波变换、字典学习、稀疏编码等。
- 重构恢复关注的是如何基于稀疏性从少量观测中恢复原信号。

- 应用：个性推荐

目录 /CONTENTS

- 对于二分类问题，给定样例集$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$，$x_i \in \mathcal{X}$，$y_i \in \mathcal{Y}$中的所有样本服从一个隐含未知的分布$\mathcal{D}$，*D*中所有样本都是独立同分布的：
  令*h*为从$\mathcal{X}$到$\mathcal{Y}$的一个映射，其泛化误差为：

$$E(h; \mathcal{D}) = P_{\boldsymbol{x} \sim \mathcal{D}}(h(x) \neq y)$$

  *h*在*D*上的经验误差为：

$$\hat{E}(h; D) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}\,(h(x_i) \neq y_i)$$

  由于*D*是$\mathcal{D}$的独立同分布采样，因此*h*的经验误差的期望等于其泛化误差。
  我们将$E(h; \mathcal{D})$和$\hat{E}(h; D)$简记为$E(h)$和$\hat{E}(h)$,令ϵ为$E(h)$的上限，即$E(h) \leq \epsilon$。
  通常用ϵ表示预先设定的学得模型所应满足的误差要求，亦称”误差参数“。

- 研究问题：经验误差与泛化误差之间的逼近程度
  若*h*在*D*上的经验误差为0，则称*h*与*D*一致，否则称其与*D*不一致。
  对任意两个映射$h_1, h_2 \in \mathcal{X} \rightarrow \mathcal{Y}$，可通过其”不合“来度量它们之间的差别：

$$\mathrm{d}(h_1, h_2) = P_{\boldsymbol{x} \sim \mathcal{D}}\big(h_1(x) \neq h_2(x)\big)$$

Chap 12 Part 2 PAC学习

- 概率近似正确学习理论（PAC）以较大的概率学得误差满足预设上限的目标。
- **PAC学习的目标**：希望以较大的概率学得误差满足预设上限的模型。
- PAC学习给出了一个刻画机器学习能力的框架。

- **目标概念**：若概念c使得$c(x) = y$成立，则称c为目标概念。
- **概念类**：所有我们希望学得的目标概念集合称为”概念类“，用符号$\mathcal{C}$表示。
- **假设空间**：给定学习算法ξ，它所考虑的所有可能概念的集合称为”假设空间“，用符号$\mathcal{H}$表示，$\mathcal{H}$和$\mathcal{C}$通常是不同的，学习算法会把自认为可能的目标概念集中起来构成$\mathcal{H}$。
  - **可分的**：若目标概念$c \in \mathcal{H}$，则$\mathcal{H}$中存在假设能将所有样示按与真实标记一致的方式完全分开，我们称该问题对学习算法ξ是可分的，亦称一致的；
  - **不可分的**：若$c \notin \mathcal{H}$，则$\mathcal{H}$中不存在任何假设能将所有示例完全正确分开的方式完全分开，我们称该问题对学习算法ξ是不可分的，亦称不一致的。

Chap 12 Part 2 PAC学习

定义 12.1 PAC 辨识 (PAC Identify): 对 $0 < \epsilon, \delta < 1$, 所有 $c \in \mathcal{C}$ 和分布 $\mathcal{D}$, 若存在学习算法 $\mathfrak{L}$, 其输出假设 $h \in \mathcal{H}$ 满足

$$P(E(h) \leqslant \epsilon) \geqslant 1 - \delta\,, \tag{12.9}$$

则称学习算法 $\mathfrak{L}$ 能从假设空间 $\mathcal{H}$ 中 PAC 辨识概念类 $\mathcal{C}$.

这样的学习算法 $\mathfrak{L}$ 能以较大的概率 (至少 $1-\delta$) 学得目标概念 c 的近似 (误差最多为 ϵ). 在此基础上可定义:

定义 12.2 PAC 可学习 (PAC Learnable): 令 m 表示从分布 $\mathcal{D}$ 中独立同分布采样得到的样例数目, $0 < \epsilon, \delta < 1$, 对所有分布 $\mathcal{D}$, 若存在学习算法 $\mathfrak{L}$ 和多项式函数 $\text{poly}(\cdot,\cdot,\cdot,\cdot)$, 使得对于任何 $m \geqslant \text{poly}(1/\epsilon, 1/\delta, \text{size}(\boldsymbol{x}), \text{size}(c))$, $\mathfrak{L}$ 能从假设空间 $\mathcal{H}$ 中 PAC 辨识概念类 $\mathcal{C}$, 则称概念类 $\mathcal{C}$ 对假设空间 $\mathcal{H}$ 而言是 PAC 可学习的, 有时也简称概念类 $\mathcal{C}$ 是 PAC 可学习的.

定义 12.3 PAC 学习算法 (PAC Learning Algorithm): 若学习算法 $\mathfrak{L}$ 使概念类 $\mathcal{C}$ 为 PAC 可学习的, 且 $\mathfrak{L}$ 的运行时间也是多项式函数 $\text{poly}(1/\epsilon, 1/\delta, \text{size}(\boldsymbol{x}), \text{size}(c))$, 则称概念类 $\mathcal{C}$ 是高效 PAC 可学习 (efficiently PAC learnable) 的, 称 $\mathfrak{L}$ 为概念类 $\mathcal{C}$ 的 PAC 学习算法.

- **关键**：假设空间$\mathcal{H}$的复杂度
  - $\mathcal{H} = \mathcal{C}$，恰PAC可学习
  - $\mathcal{H} \neq \mathcal{C}$，$\mathcal{H}$越大，包含任意目标概念的可能性越大，但从中找到某个具体目标概念的难度也越大。
  - $|\mathcal{H}|$有限时，称为有限假设空间，否则称为无限假设空间。

- **可分情形**

  可分情形意味着目标概念c属于假设空间$\mathcal{H}$，即$c \in \mathcal{H}$。

- **学习策略**：只保留与D一致的假设，剔除与D不一致的假设。若训练集足够大，则可不断借助D中的样例剔除不一致的假设，直到$\mathcal{H}$中仅剩下一个假设为止，这个假设就是目标概念c。
- **缺点**：无法进一步区分等效假设的优劣。
- **训练集D的规模**：对于PAC学习来说，只要训练集D的规模能使学习算法ξ能以概率$1-\delta$找到目标假设的ϵ近似即可。

$$m \geqslant \frac{1}{\epsilon}\left(\ln|\mathcal{H}| + \ln\frac{1}{\delta}\right). \tag{12.14}$$

由此可知，有限假设空间 $\mathcal{H}$ 都是 PAC 可学习的，所需的样例数目如式(12.14)所示，输出假设 h 的泛化误差随样例数目的增多而收敛到 0，收敛速率为 $O(\frac{1}{m})$.

- **不可分情形**

  目标概念c不存在于假设空间$\mathcal{H}$中，$\mathcal{H}$中的任意一个假设都会在训练集上出现或多或少的错误。

- 当样例数目m较大时，h的经验误差是其泛化误差的很好的近似。
- **结论**
- **不可知学习**：当$c \notin \mathcal{H}$时，学习算法ξ无法学得目标概念c的ϵ近似，但是，当假设空间$\mathcal{H}$给定时，其中必存在一个泛化误差最小的假设，目标是寻找此假设的ϵ近似。

定义 12.5 不可知 PAC 可学习 (agnostic PAC learnable): 令 m 表示从分布 $\mathcal{D}$ 中独立同分布采样得到的样例数目, $0 < \epsilon, \delta < 1$, 对所有分布 $\mathcal{D}$, 若存在学习算法 $\mathfrak{L}$ 和多项式函数 $\text{poly}(\cdot,\cdot,\cdot,\cdot)$, 使得对于任何 $m \geqslant \text{poly}(1/\epsilon, 1/\delta, \text{size}(\boldsymbol{x}), \text{size}(c))$, $\mathfrak{L}$ 能从假设空间 $\mathcal{H}$ 中输出满足式(12.20)的假设 h:

$$P\left(E(h) - \min_{h' \in \mathcal{H}} E(h') \leqslant \epsilon\right) \geqslant 1 - \delta\,, \tag{12.20}$$

则称假设空间 $\mathcal{H}$ 是不可知 PAC 可学习的.

Chap 11 Part 4 VC维

- 现实学习任务通常是无限假设空间，需度量假设空间的复杂度，常见的办法是考虑假设空间的VC维。
- **增长函数**描述了空间$\mathcal{H}$的表示能力，反映了假设空间的复杂度。

定义 12.6 对所有 $m \in \mathbb{N}$, 假设空间 $\mathcal{H}$ 的增长函数 $\Pi_{\mathcal{H}}(m)$ 为

$$\Pi_{\mathcal{H}}(m) = \max_{\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m\}\subseteq\mathcal{X}} \left|\left\{\left(h\left(\boldsymbol{x}_1\right),\ldots,h\left(\boldsymbol{x}_m\right)\right) \mid h \in \mathcal{H}\right\}\right| . \tag{12.21}$$

增长函数 $\Pi_{\mathcal{H}}(m)$ 表示假设空间 $\mathcal{H}$ 对 m 个示例所能赋予标记的最大可能结果数. 显然, $\mathcal{H}$ 对示例所能赋予标记的可能结果数越大, $\mathcal{H}$ 的表示能力越强, 对学习任务的适应能力也越强. 因此, 增长函数描述了假设空间 $\mathcal{H}$ 的表示能力, 由此反映出假设空间的复杂度. 我们可利用增长函数来估计经验误差与泛化误差之间的关系:

Chap 11 Part 4 VC维

定理 12.2 对假设空间 $\mathcal{H}$, $m \in \mathbb{N}$, $0 < \epsilon < 1$ 和任意 $h \in \mathcal{H}$ 有

$$P\left(\left|E(h) - \widehat{E}(h)\right| > \epsilon\right) \leqslant 4\Pi_{\mathcal{H}}(2m)\exp\left(-\frac{m\epsilon^2}{8}\right). \tag{12.22}$$

假设空间 $\mathcal{H}$ 中不同的假设对于 D 中示例赋予标记的结果可能相同, 也可能不同; 尽管 $\mathcal{H}$ 可能包含无穷多个假设, 但其对 D 中示例赋予标记的可能结果数是有限的: 对 m 个示例, 最多有 2^m 个可能结果. 对二分类问题来说, $\mathcal{H}$ 中的假设对 D 中示例赋予标记的每种可能结果称为对 D 的一种“对分”. 若假设空间 $\mathcal{H}$ 能实现示例集 D 上的所有对分, 即 $\Pi_{\mathcal{H}}(m) = 2^m$, 则称示例集 D 能被假设空间 $\mathcal{H}$ “打散”.

- **对分和打散的概念**

- **VC维的计算**

通常这样来计算 $\mathcal{H}$ 的 VC 维: 若存在大小为 d 的示例集能被 $\mathcal{H}$ 打散, 但不存在任何大小为 $d+1$ 的示例集能被 $\mathcal{H}$ 打散, 则 $\mathcal{H}$ 的 VC 维是 d. 下面给出两

定义 12.7 假设空间 $\mathcal{H}$ 的 VC 维是能被 $\mathcal{H}$ 打散的最大示例集的大小, 即

$$\mathrm{VC}(\mathcal{H}) = \max\{m : \Pi_{\mathcal{H}}(m) = 2^m\}. \tag{12.23}$$

$\mathrm{C}(\mathcal{H}) = d$ 表明存在大小为 d 的示例集能被假设空间 $\mathcal{H}$ 打散. 注意: 这并不意味着所有大小为 d 的示例集都能被假设空间 $\mathcal{H}$ 打散. 细心的读者可能已发现, VC 维的定义与数据分布 $\mathcal{D}$ 无关! 因此, 在数据分布未知时仍能计算出假设空间 $\mathcal{H}$ 的 VC 维.

结论：基于VC维的泛化误差界是分布无关、数据独立的。
任何VC维有限的假设空间 $\mathcal{H}$ 都是（不可知）PAC可学习的。

Chap 11 Part 5 Rademacher复杂度

- Rademacher复杂度用来刻画假设空间的复杂度，和VC维不同的是，它在一定程度上考虑了数据分布，通常比基于VC维的泛化误差界更紧一些。
- 从Rademacher复杂度和增长函数可以推导出基于VC维的泛化误差界。
- VC维和Rademacher复杂度所得到的结果均与具体学习算法无关，对所有学习算法都适用。

Chap 11 Part 6 稳定性

- 算法的稳定性考察的是在算法输入变化时，输出是否会随之发生较大的变化。
- **稳定性分析**不必考虑假设空间中所有可能的假设，只需根据算法自身的特性（稳定性）来讨论输出假设的泛化误差界。

损失函数 $\ell(\mathfrak{L}_D(\boldsymbol{x}), y) : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ 刻画了假设 $\mathfrak{L}_D$ 的预测标记 $\mathfrak{L}_D(\boldsymbol{x})$ 与真实标记 y 之间的差别, 简记为 $\ell(\mathfrak{L}_D, \boldsymbol{z})$. 下面定义关于假设 $\mathfrak{L}_D$ 的几种损失.

- 泛化损失

$$\ell(\mathfrak{L}, \mathcal{D}) = \mathbb{E}_{\boldsymbol{x}\in\mathcal{X}, \boldsymbol{z}=(\boldsymbol{x},y)}\left[\ell(\mathfrak{L}_D, \boldsymbol{z})\right] . \tag{12.54}$$

- 经验损失

$$\widehat{\ell}(\mathfrak{L}, D) = \frac{1}{m}\sum_{i=1}^{m}\ell(\mathfrak{L}_D, \boldsymbol{z}_i) . \tag{12.55}$$

- 留一(leave-one-out)损失

$$\ell_{loo}(\mathfrak{L}, D) = \frac{1}{m}\sum_{i=1}^{m}\ell(\mathfrak{L}_{D^{\setminus i}}, \boldsymbol{z}_i) . \tag{12.56}$$

定义 12.10 对任何 $\boldsymbol{x} \in \mathcal{X}$, $\boldsymbol{z} = (\boldsymbol{x}, y)$, 若学习算法 $\mathfrak{L}$ 满足

$$\left|\ell(\mathfrak{L}_D, \boldsymbol{z}) - \ell(\mathfrak{L}_{D^{\setminus i}}, \boldsymbol{z})\right| \leqslant \beta , \quad i = 1, 2, \ldots, m,$$

则称 $\mathfrak{L}$ 关于损失函数 ℓ 满足 β-均匀稳定性.

《TensorFlow实战Google深度学习框架》

第三章 TensorFlow入门

目录 /CONTENTS

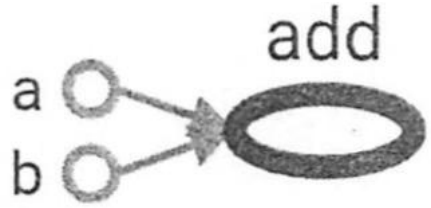
Chap 3

TensorFlow入门

- 计算图
- 张量
- 会话
- TensorFlow实现神经网络

计算图

- 计算图是一个有向图，是对TensorFlow中计算任务的抽象描述，也称为数据流图。TensorFlow使用计算图将计算表示成了独立的指令之间的依赖关系，在计算图中，节点表示计算单元（即一个独立的运算操作），图中的边表示计算使用或产生的数据。在TensorFlow1.x版本中采用的是静态图机制，我们需要预先定义好计算图，然后再可以反复的调用它。 （补充：目前TensorFlow已更新至2.0版本， TensorFlow2.0则采用了动态图机制，我们可以像执行普通的python程序一样执行TensorFlow的代码，而不再需要自己预先定义好静态图，调试代码也更加容易。）



计算图

```python
import tensorflow as tf
g1 = tf.Graph()
with g1.as_default():
    v = tf.get_variable("v", shape=[1],initializer=tf.zeros_initializer)#在计算图g1中定义变量"v"，并设置初始值为0。

g2 = tf.Graph()
with g2.as_default():
    v = tf.get_variable("v", shape=[1],initializer=tf.ones_initializer)#在计算图g1中定义变量"v"，并设置初始值为0。

# 计算图g1中读取变量"v"的取值
with tf.Session(graph=g1) as sess:
    tf.initialize_all_variables().run()
    with tf.variable_scope("",reuse=True):
        print(sess.run(tf.get_variable("v")))

# 计算图g2中读取变量"v"的取值
with tf.Session(graph=g2) as sess:
    tf.initialize_all_variables().run()
    with tf.variable_scope("",reuse=True):
        print(sess.run(tf.get_variable("v")))
```

- TensorFlow通过tf.Graph函数来生成新的计算图，不同计算图上的张量和运算不会共享。

```
[0.]
[1.]

Process finished with exit code 0
```

张量

- 从功能角度上看，张量可以被简单理解为多维数组。其中零阶张量表示标量；一阶张量为向量即一维数组；第n阶向量可以理解为一个n维数组。

```
import tensorflow as tf
#tf.constant是一个计算，这个计算的结果为一个张量，保存在变量a中。
a = tf.constant([1.0,2.0],name="a")
b = tf.constant([2.0,3.0],name="b")
result = tf.add(a,b,name="add")
print(result)
```

- 输出为：

```
Tensor("add:0", shape=(2,), dtype=float32)

Process finished with exit code 0
```

- 一个张量主要保存三个属性：名字（name）、维度（shape）和类型（type）。

1. 名字属性是张量的唯一标识符。张量的命名就可以通过”node:src_output“的形式给出。其中node为节点的名称，src_output表示当前张量来自节点的第几个输出。
2. 维度属性描述了一个张量的维度信息。
3. 类型属性dtype表示张量的类型信息。

会话

- TensorFlow中的会话（session）来执行定义好的运算。会话拥有并管理TensorFlow程序运行时的所有资源。当所有计算完成之后需要关闭会话来帮助系统回收资源，否则就可能出现资源泄漏的问题。

```
import tensorflow as tf
#tf.constant是一个计算，这个计算的结果为一个张量，保存在变量a中。
a = tf.constant([1.0,2.0],name="a")
b = tf.constant([2.0,3.0],name="b")
result = tf.add(a,b,name="add")
sess = tf.Session()#创建一个会话
sess.run(result)#获得张量result的取值
print(sess.run(result))
sess.close()#关闭会话释放资源
```

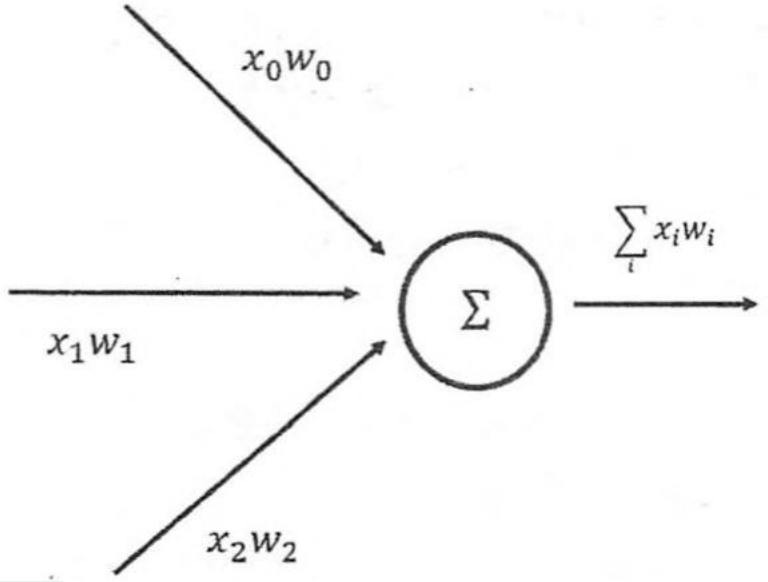
```
[3. 5.]

Process finished with exit code 0
```

- TensorFlow也可以通过python的上下文管理器来使用会话。通过Python上下文管理器的机制，只要将所有的计算放在“with”的内部就可以。当上下文管理器退出的时候会自动释放所有资源。

```
import tensorflow as tf
#tf.constant是一个计算，这个计算的结果为一个张量，保存在变量a中。
a = tf.constant([1.0,2.0],name="a")
b = tf.constant([2.0,3.0],name="b")
result = tf.add(a,b,name="add")
with tf.Session() as sess:
    sess.run(result)
    print(sess.run(result))
```

TensorFlow实现神经网络

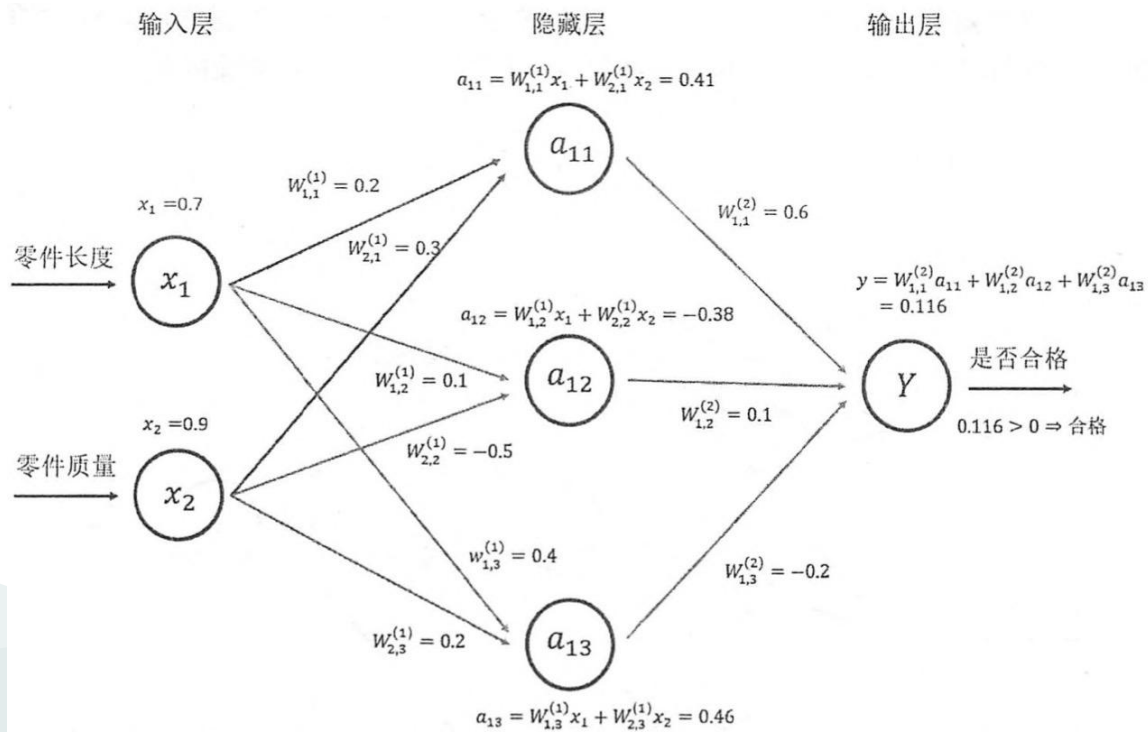
- 神经元是构成神经网络的最小单元。一个神经元有多个输入和一个输出。每个神经元的输入既可以是其他神经元的输出，也可以是整个神经网络的输入。所谓神经网络的结构就是指不同神经元之间的连接结构。如下图，一个最简单的神经元结构的输出就是所有输入的加权和，而不同输入的权重就是神经元的参数。神经网络的优化过程就是优化神经元中参数的取值的过程。



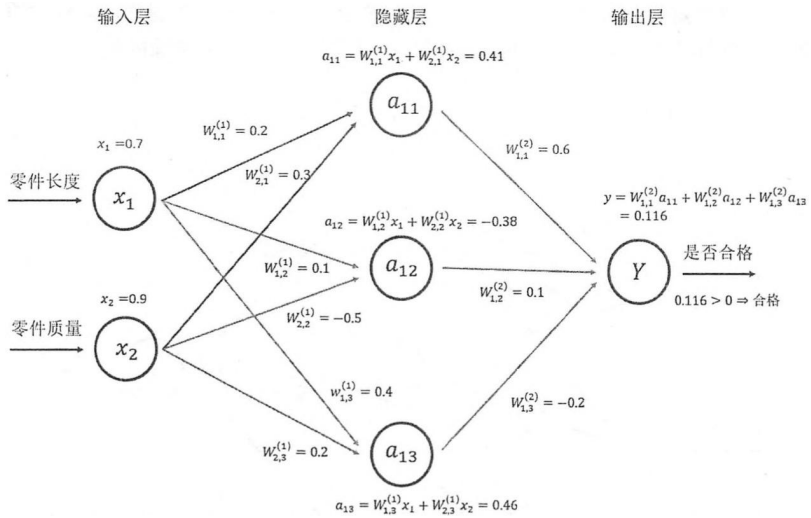
前向传播算法

-全连接网络结构的前向传播算法。

- 全连接神经网络：相邻两层之间任意两个节点之间都有连接。
- 计算神经网络前向传播结果需要三部分信息。第一个部分是神经网络的输入，这个输入就是从实体中提取的特征向量。第二部分为神经网络的连接结构，神经网络是由神经元构成的，神经网络的结构给出不同神经元之间输入输出的连接关系。第三部分为每个神经元中的参数。



前向传播算法



- 前向传播算法可以表示为矩阵乘法。将输入x1.x2组织成一个1x2的矩阵x=[x1,x2]，而W(1)组织成一个2x3的矩阵：

$$W^{(1)}=\begin{bmatrix} W_{1,1}^{(1)} & W_{1,2}^{(1)} & W_{1,3}^{(1)} \\ W_{2,1}^{(1)} & W_{2,2}^{(1)} & W_{2,3}^{(1)} \end{bmatrix}$$

- 这样通过矩阵乘法可以得到隐藏层三个节点所组成的向量取值：

$$a^{(1)}=[a_{11},a_{12},a_{13}]=xW^{(1)}=[x_1,x_2]\begin{bmatrix} W_{1,1}^{(1)} & W_{1,2}^{(1)} & W_{1,3}^{(1)} \\ W_{2,1}^{(1)} & W_{2,2}^{(1)} & W_{2,3}^{(1)} \end{bmatrix}$$

$$=[W_{1,1}^{(1)}x_1+W_{2,1}^{(1)}x_2,W_{1,2}^{(1)}x_1+W_{2,2}^{(1)}x_2,W_{1,3}^{(1)}x_1+W_{2,3}^{(1)}x_2]$$

- 类似的输出层可以表示为：

$$[y]=a^{(1)}W^{(2)}=[a_{11},a_{12},a_{13}]\begin{bmatrix} W_{1,1}^{(2)} \\ W_{2,1}^{(2)} \\ W_{3,1}^{(2)} \end{bmatrix}=[W_{1,1}^{(2)}a_{11}+W_{1,2}^{(2)}a_{12}+W_{1,3}^{(2)}a_{13}]$$

前向传播算法

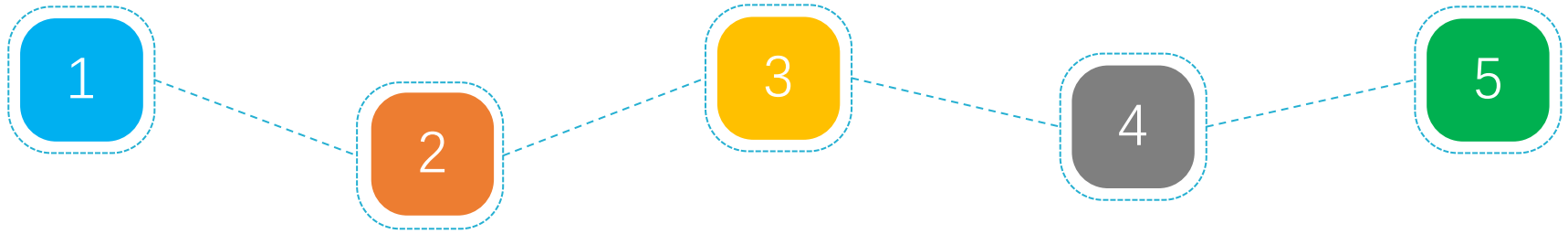
- 使用TensorFlow程序实现该前向传播过程：

```
import tensorflow as tf
#声明w1、w2两个变量。令seed=1，保证每次运行结果都是一致的。
w1 = tf.Variable(tf.random_normal([2,3],stddev=1,seed=1))
w2 = tf.Variable(tf.random_normal([3,1],stddev=1,seed=1))
#输入特征向量定义为一个常量，且为 1 * 2的矩阵。
x = tf.constant([[0.7,0.9]])
#前向传播算法，matmul进行矩阵相乘
a = tf.matmul(x,w1)
y = tf.matmul(a,w2)

sess = tf.Session()

#初始化w1和w2
sess.run(w1.initializer)
sess.run(w2.initializer)
print(sess.run(y))
sess.close()
```

```
[[3.957578]]

Process finished with exit code 0
```

DeepSigns

目录
DIRECTORY



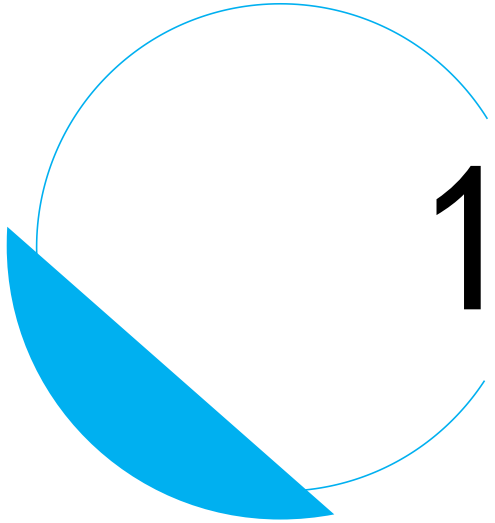
研究背景

DeepSigns方法

白盒模块

黑盒模块

算法性能



研究背景

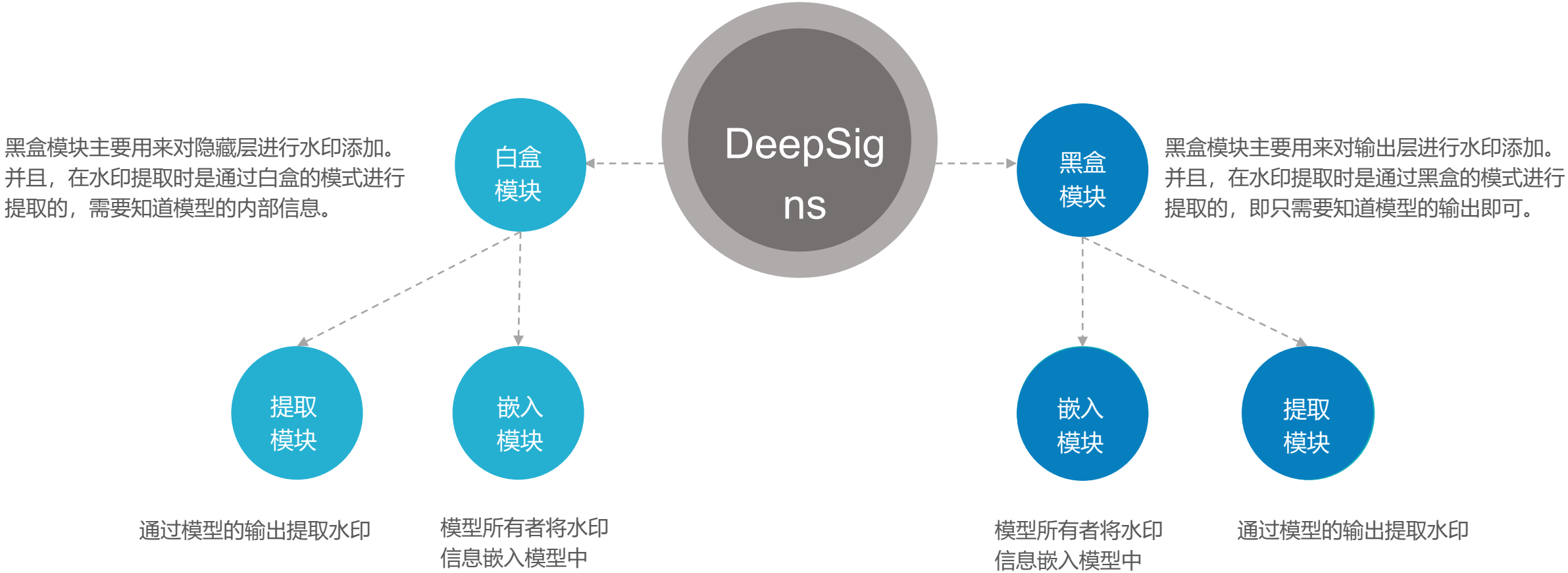
模型训练花费巨大，模型容易被窃取

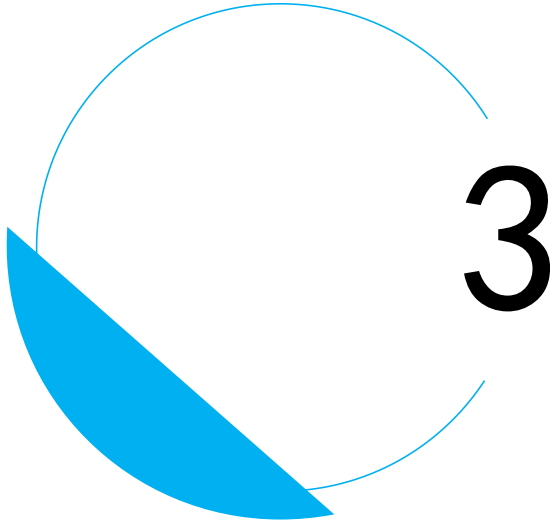
虽然现在深度学习模型被应用在各个方面。但是，一个能够被利用的模型往往需要巨大的训练集和很长的训练时间。这些花费都是巨大的。所以，一个可以使用的深度学习模型的价值是巨大的。可以当做一个知识产权来处理。所以需要对模型的知识产权进行保护。

2

DeepSigns方法

DeepSigns


DeepSigns
白盒模块
黑盒模块
提取模块
嵌入模块
嵌入模块
提取模块
黑盒模块主要用来对隐藏层进行水印添加。并且，在水印提取时是通过白盒的模式进行提取的，需要知道模型的内部信息。
黑盒模块主要用来对输出层进行水印添加。并且，在水印提取时是通过黑盒的模式进行提取的，即只需要知道模型的输出即可。
通过模型的输出提取水印
模型所有者将水印信息嵌入模型中
模型所有者将水印信息嵌入模型中
通过模型的输出提取水印



白盒模块

DeepSigns

水印嵌入流程

S：DNN的分类数量。
T：一种标签。
X：选取的标签为T的训练数据集
A：生成的投影矩阵
M：隐藏层输出的第二维的值
N：水印长度

Algorithm 1 Watermark embedding for one hidden layer.

INPUT: **Topology of the unmarked DNN** $(\mathscr{T})$; **Training data** $(\{X^{train}, Y^{train}\})$; **Owner-specific watermark signature b; Total number of Gaussian classes** (S); **Length of watermark vector for each selected distribution** (N); **Dimensionality of the activation map in the embedded layer** (M); **and Embedding strength hyper-parameters** (λ_1, λ_2).

OUTPUT: **Watermarked DNN** $(\mathscr{T}^*)$; **WM keys.**

❶ Key Generation:

$$T \leftarrow Select\_Gaussian\_Classes\ ([1, S])$$
$$X^{key} \leftarrow Subset\_Training\_Data\ (T, \{X^{train}, Y^{train}\})$$
$$A^{M\times N} \leftarrow Generate\_Secret\_Matrix\ (M, N)$$

❷ Model Fine-tuning: Two additive regularization loss functions are incorporated to train the DNN:

$$L = \underbrace{cross\_entropy}_{loss_0} + \lambda_1 loss_1 + \lambda_2 loss_2.$$

Return: Marked DNN $\mathscr{T}^*$, WM keys $(s, X^{key}, A^{M\times N})$.

DeepSigns

水印嵌入与提取算法

$$G_{\sigma}^{s\times N} = Sigmoid\left(\mu_{l}^{s\times M} \cdot A^{M\times N}\right),$$
$$\tilde{b}^{s\times N} = Hard\_Thresholding\left(G_{\sigma}^{s\times N}, 0.5\right).$$

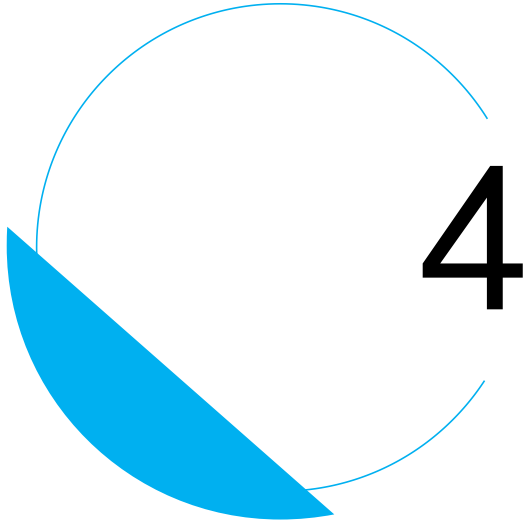
μ：选取的隐藏层的高斯分布
b：嵌入的水印
Hard_thresholding()函数将小于0.5的值变为0，
大于0.5的值变为1

通过模型微调，将μ变为能够提取水印的值

❷ **Model Fine-tuning.** To effectively encode the WM information, two additional constraints need to be considered during DNN training: (i) Selected activations shall be isolated from other activations. (ii) The distance between the owner-specific WM signature and the transformation of isolated activations shall be minimal. We design and incorporate two specific loss functions to address each of these two constraints ($loss_1$ and $loss_2$ in Step 2 of Algorithm 1).

$$\lambda_1 \underbrace{\left(\Sigma_{i\in T}\|\mu_l^i - f_l^i(x,\theta)\|_2^2 - \Sigma_{i\in T, j\notin T}\|\mu_l^i - \mu_l^j\|_2^2\right)}_{\mathbf{loss_1}}.$$

$$\underbrace{-\lambda_2 \sum_{j=1}^{N}\sum_{k=1}^{s}\left(b^{kj}\ln(G_{\sigma}^{kj}) + (1-b^{kj})\ln(1-G_{\sigma}^{kj})\right)}_{\text{交叉熵 } \mathbf{loss_2}}.$$

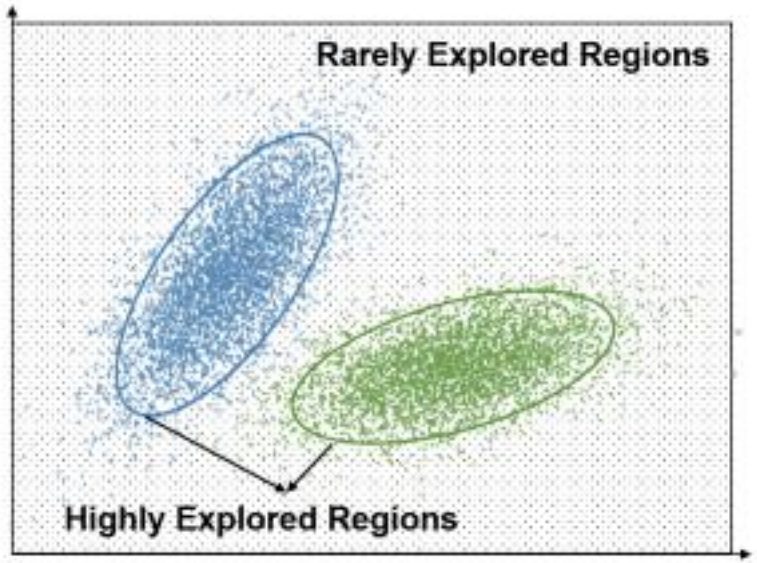


4

黑盒模块

DeepSigns

主要思想：把所有样本看成一个空间，每一个样本看做空间中的一个点。
训练样本所在的区域被看做是已经探索过的区域。这些区域保证了模型的正确性。在添加水印时使用未探索的区域。



layers in the underlying DNN. The acquired pdf, in turn, gives us an insight into both the regions that are thoroughly occupied by the training data and the regions that are only covered by a few inputs, which we refer to as rarely explored regions. Figure 4 illustrates a simple example of two clustered activation distributions spreading in a two-dimensional subspace. The procedure of WM embedding in the output layer is summarized in Algorithm 2. In the following, we explicitly discuss each of the steps outlined in Algorithm 2.

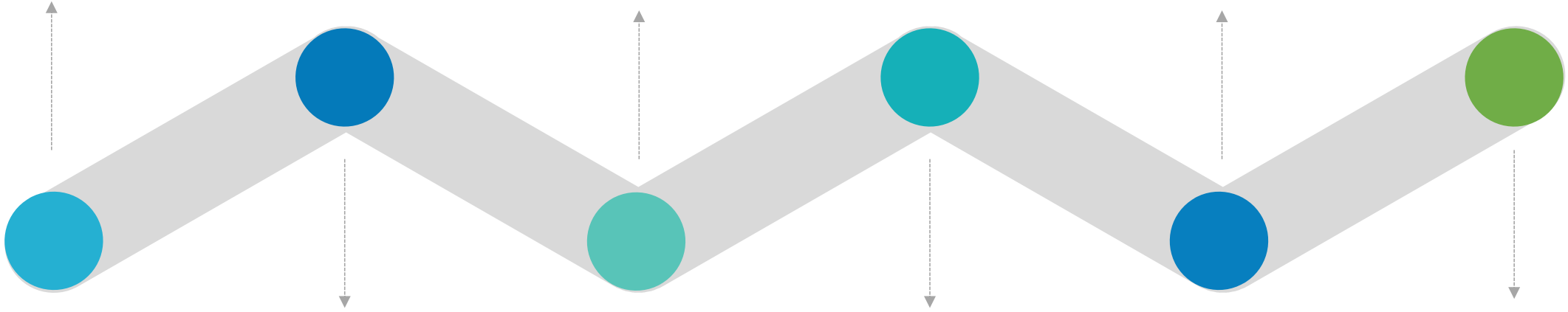
Figure 4: Due to the high dimensionality of DNNs and limited access to labeled data (the blue and green dots in the figure), there are regions within the DNN model that are rarely explored. DeepSigns exploits this mainly unused regions for WM embedding while minimally affecting the accuracy.

DeepSigns

生成n*key_len(密钥长度)张像素点为随机值图片，并为他们其分配随机标签。记为S。

将S与初始训练集一起当做训练集，对初始模型进行训练。使模型对S中的图片判断正确的张数达到一定阈值。被判断正确的S中的图片集合记为S2。

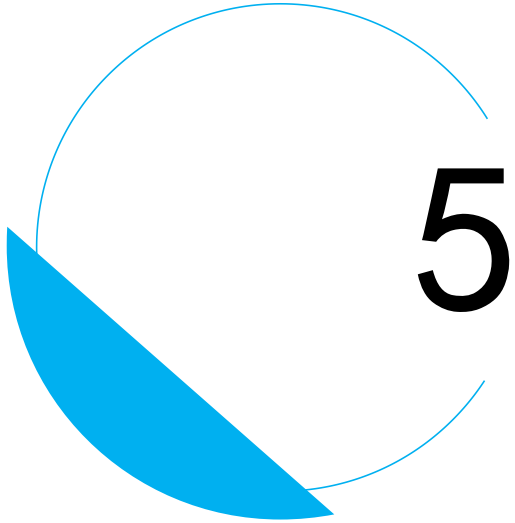
嵌入← →提取



利用未添加水印的初始模型对S中的所有图片进行分类，分类错误的图片集合记为S1。

提取S1与S2长度为key_len的交集K，并附带K中所有图片的标签作为密钥K。

将K中图片输入被检测模型，若模型的判断正确率达到一定数值，说明模型中添加了我们的水印。



算法性能

DeepSigns

水印评价指标

1、Fidelity：目标神经网络的精度不会因为水印嵌入而降低。
2、Reliability：水印提取应产生最小的误报率；应使用相关密钥能够有效检测水印。
3、Robustness：嵌入的水印应能抵抗模型微小的修改，如修剪、微调或水印覆盖。
4、Integrity：水印提取的误差较小；水印模型应使用相关密钥进行唯一标识。
5、Capacity：水印方法应能在目标DNN中嵌入大量信息.
6、Efficiency：水印嵌入和提取的通信和计算开销应该足够的小。
7、Security：水印应能抵御暴力攻击，且在目标神经网络中不留下可见痕迹；因此，未经授权的一方无法检测/移除水印的存在。

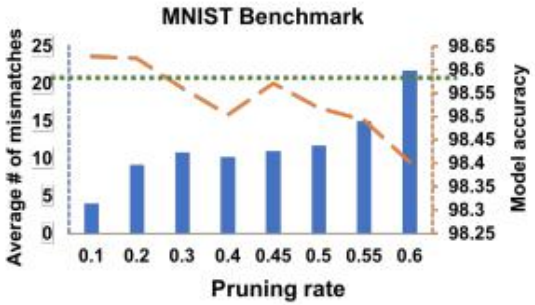
Fidelity

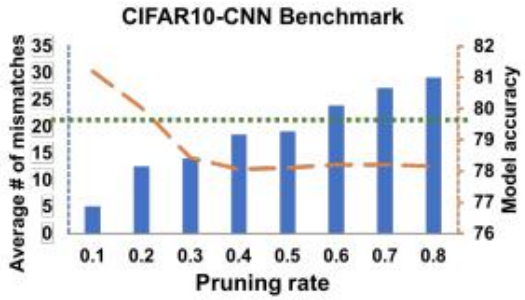
Dataset	Baseline Accuracy	Accuracy of Marked Model		DL Model Type	DL Model Architecture
MNIST	98.54%	K = 20	N = 4	MLP	784-512FC-512FC-10FC
		98.59%	98.13%		
CIFAR10	78.47%	K = 20	N = 4	CNN	3*32*32-32C3(1)-32C3(1)-MP2(1) -64C3(1)-64C3(1)-MP2(1)-512FC-10FC
		81.46%	80.70%		
CIFAR10	91.42%	K = 20	N = 128	WideResNet	Please refer to [32].
		91.48%	92.02%		
ImageNet	74.72%	K = 20	–	ResNet50	Please refer to [4].
		74.21%	–		

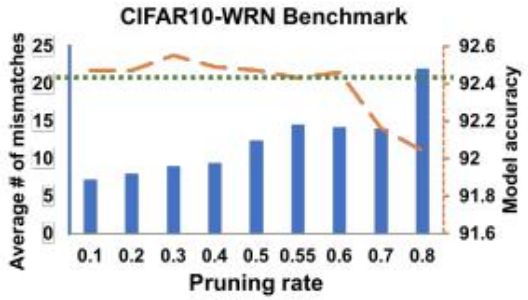
DeepSigns

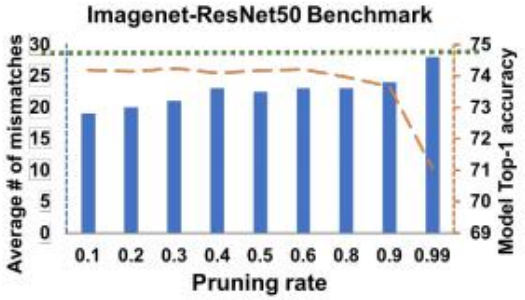
Reliability and Robustness

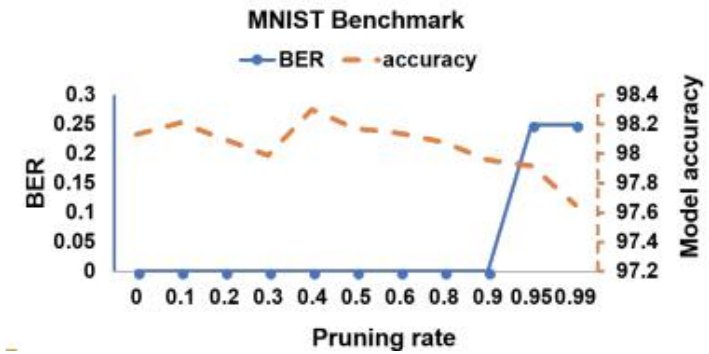
Parameter Pruning

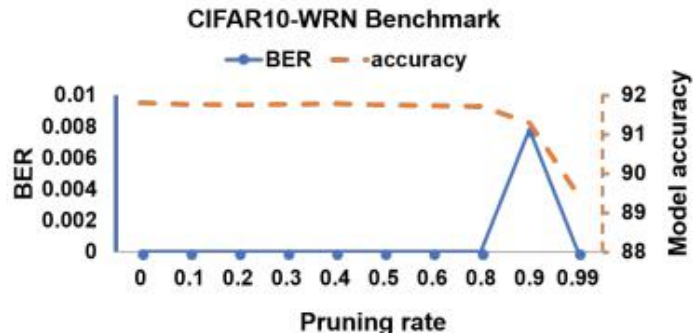


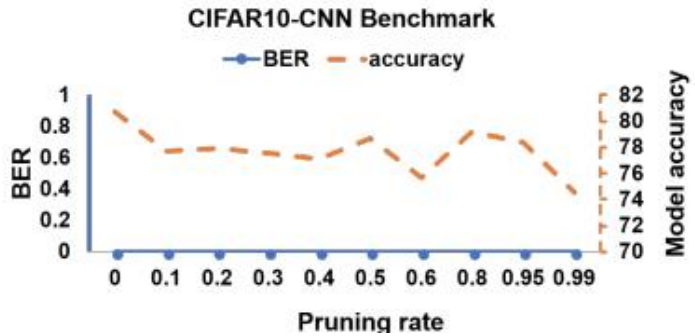






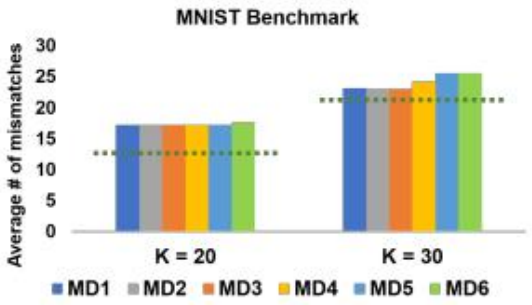


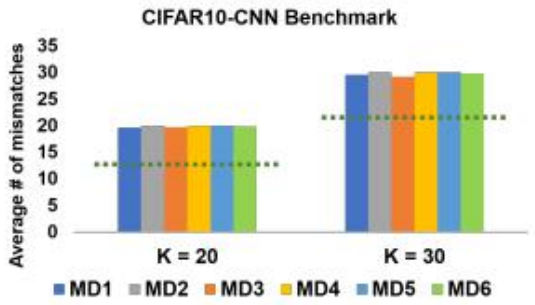


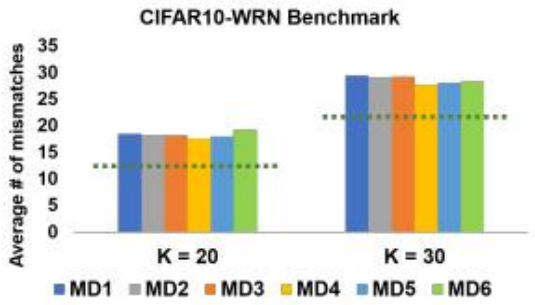


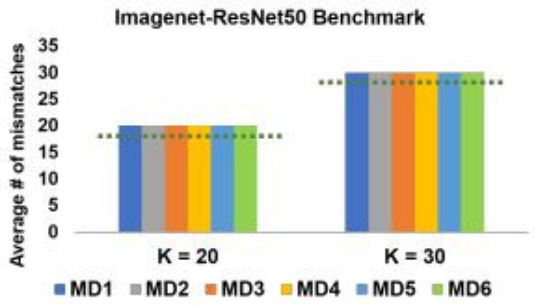
DeepSigns

Integrity





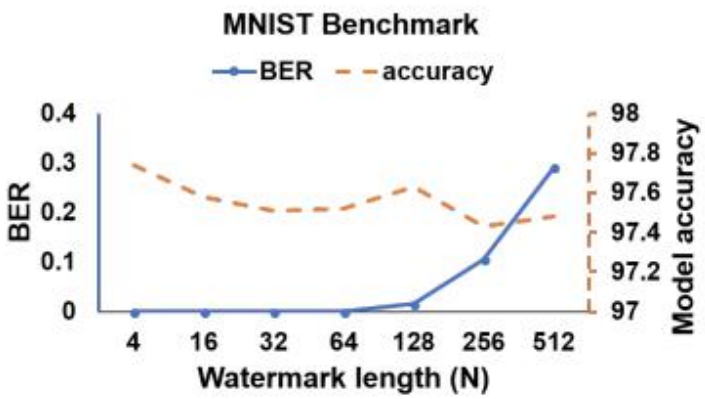


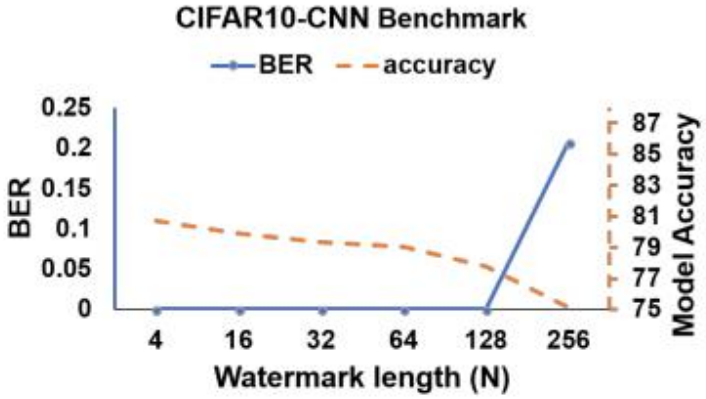


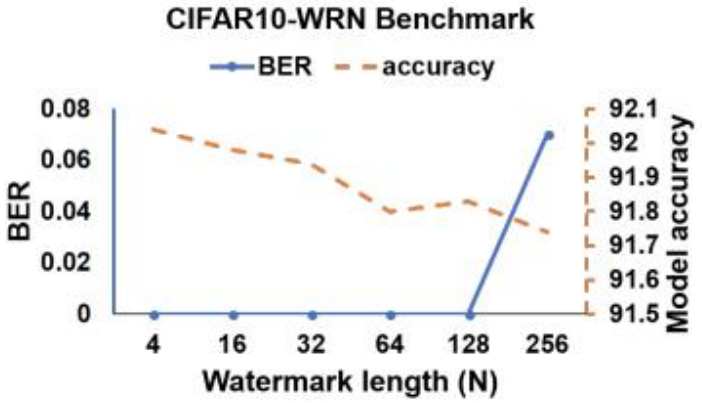
不同基准的完整性分析。绿色虚线表示不同WM长度的检测阈值。前三种模型（MD 1-3）是拓扑结构相同但参数不同的神经网络。最后三个模型（MD 4-6）是具有不同拓扑结构的神经网络（[29]、[17]、[32]）。

DeepSigns

Capacity



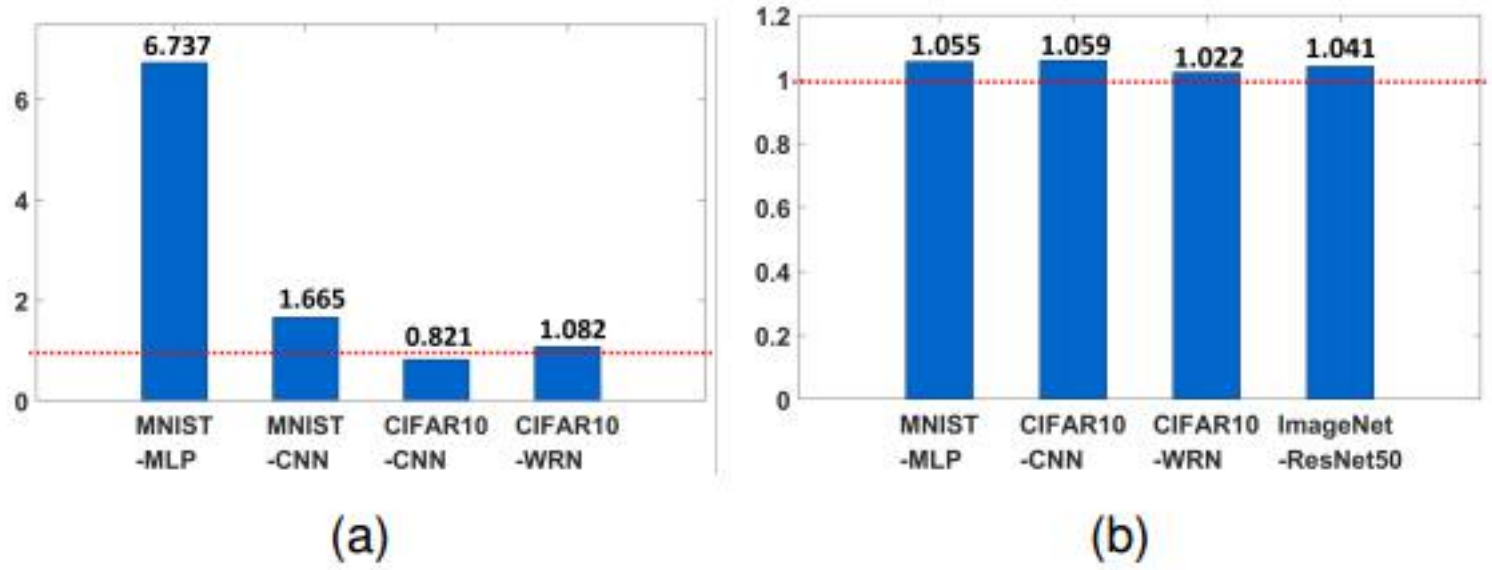




WM签名的长度（容量）和水印提取的误比特率之间存在一个折衷。随着嵌入比特数（N）的增加，水印模型的测试精度降低，WM提取的误码率增加。这一趋势表明，嵌入过多的WM信息会降低系统的保真度和可靠性。

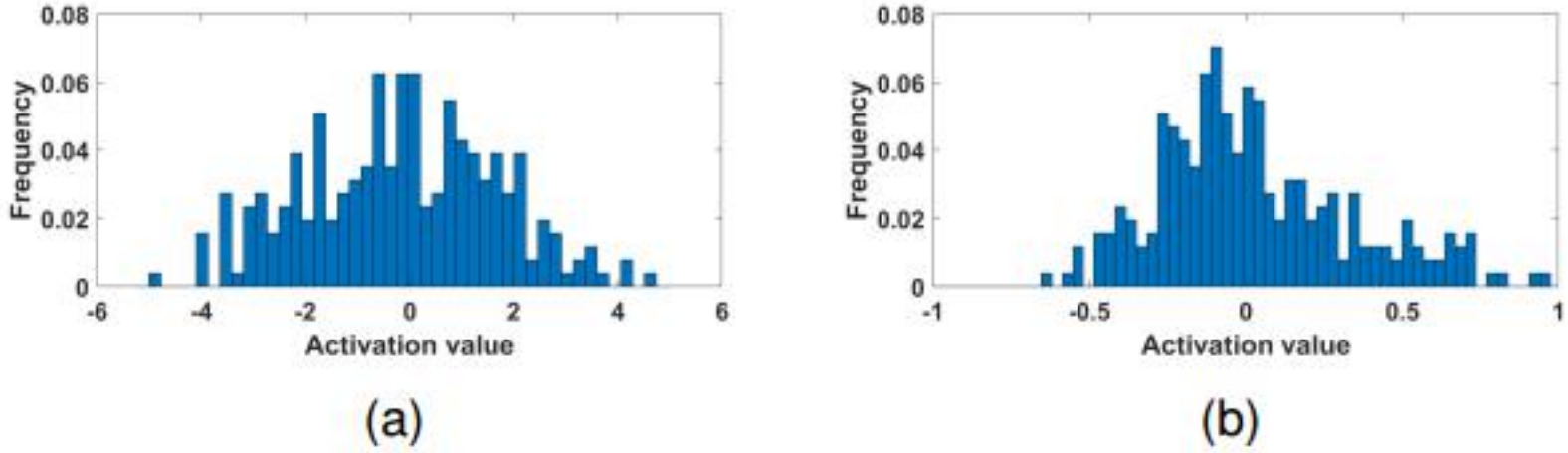
DeepSigns

Efficiency



DeepSigns

Security



（a）有标记和（b）无标记模型的激活图的分布。DeepSigns在安全嵌入WM信息的同时保留了内在分布

Deep Residual Network for Steganalysis of Digital Images

残差网络+隐写分析

2019-12-04

目录
DIRECTORY

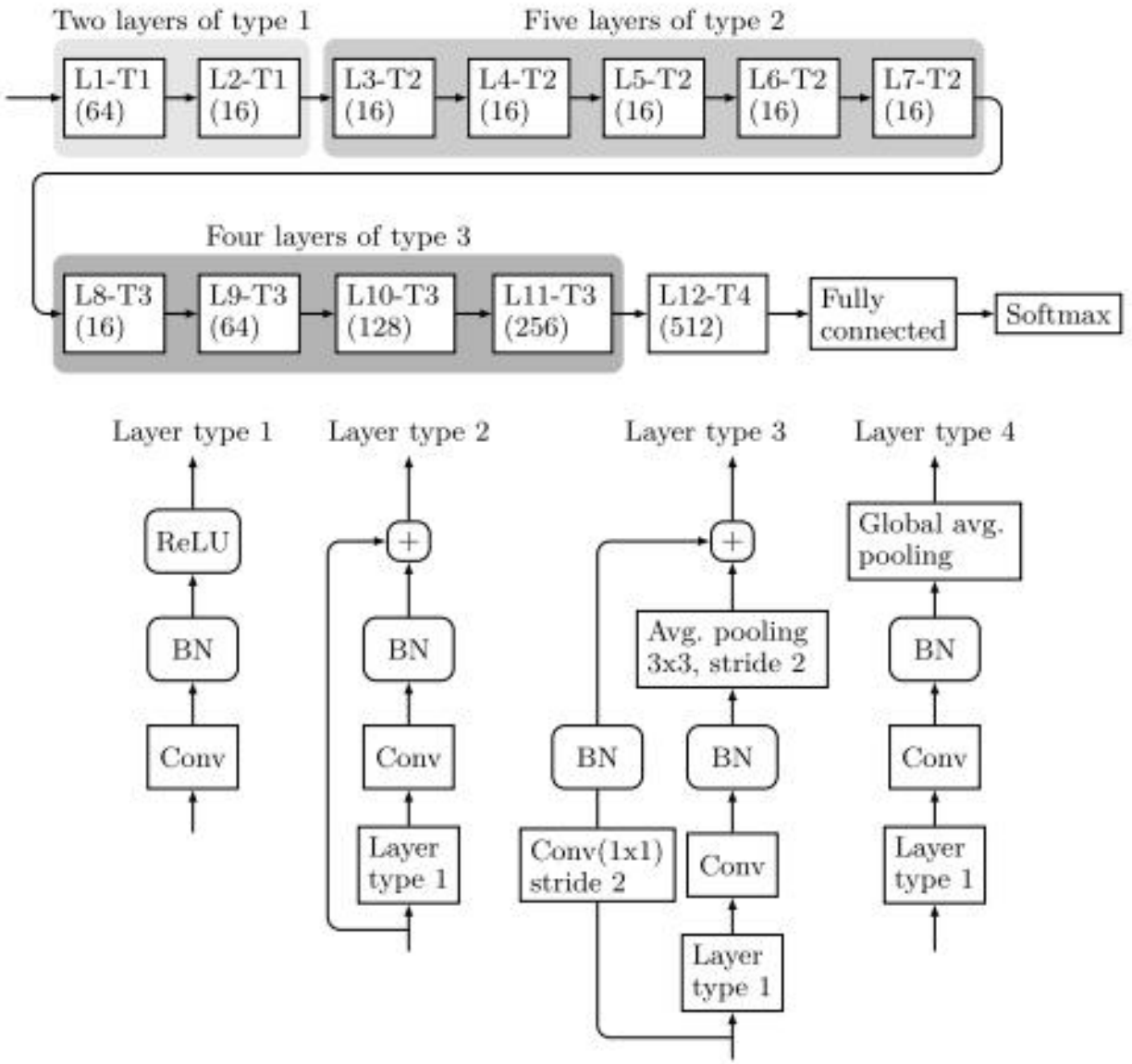
研究背景和目的

目前隐写分析模型都会引入许多外部设计因素，例如初始化卷积核、阈值化、量化。
但是初始化卷积核可能对网络性能产生不利影响，例如高通滤波器可能会抑制JPEG隐写算法引入的隐写信号。

因此，文章描述了一种深度残差架构，用于减少启发式和外部元素的使用，并且适用于空间域和JPEG域。

深度残差架构中的主要元素是残差捷径（residual shortcuts），此残差与图像预处理中的残差概念有所不同。
noise residual：隐写分析中的像素预测错误
residual layer/module/connection：深度学习中的残差网络架构

深度残差网络架构



SRNet从功能上划分，网络主要由三个部分构成。

Layers 1-7：提取噪声残差

Layers 8-12：减小特征图维度

FC + softmax：线性分类器

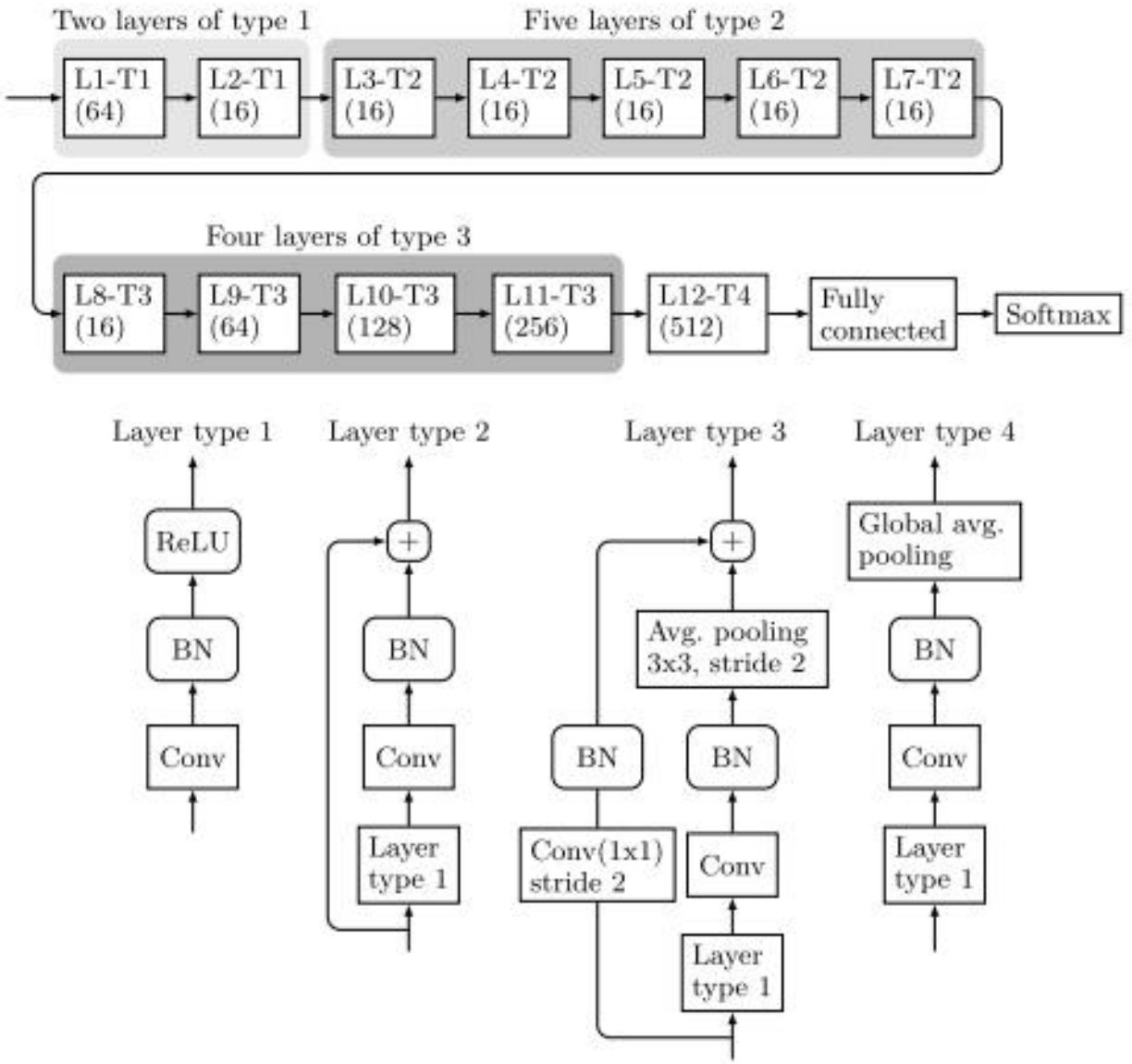
L1代表层数

T1代表层的类型

(16)卷积核的数量

所有卷积层的卷积核大小为3×3，所有非线性激活函数为ReLU，网络中的所有滤波器均随机初始化。

深度残差网络架构—续



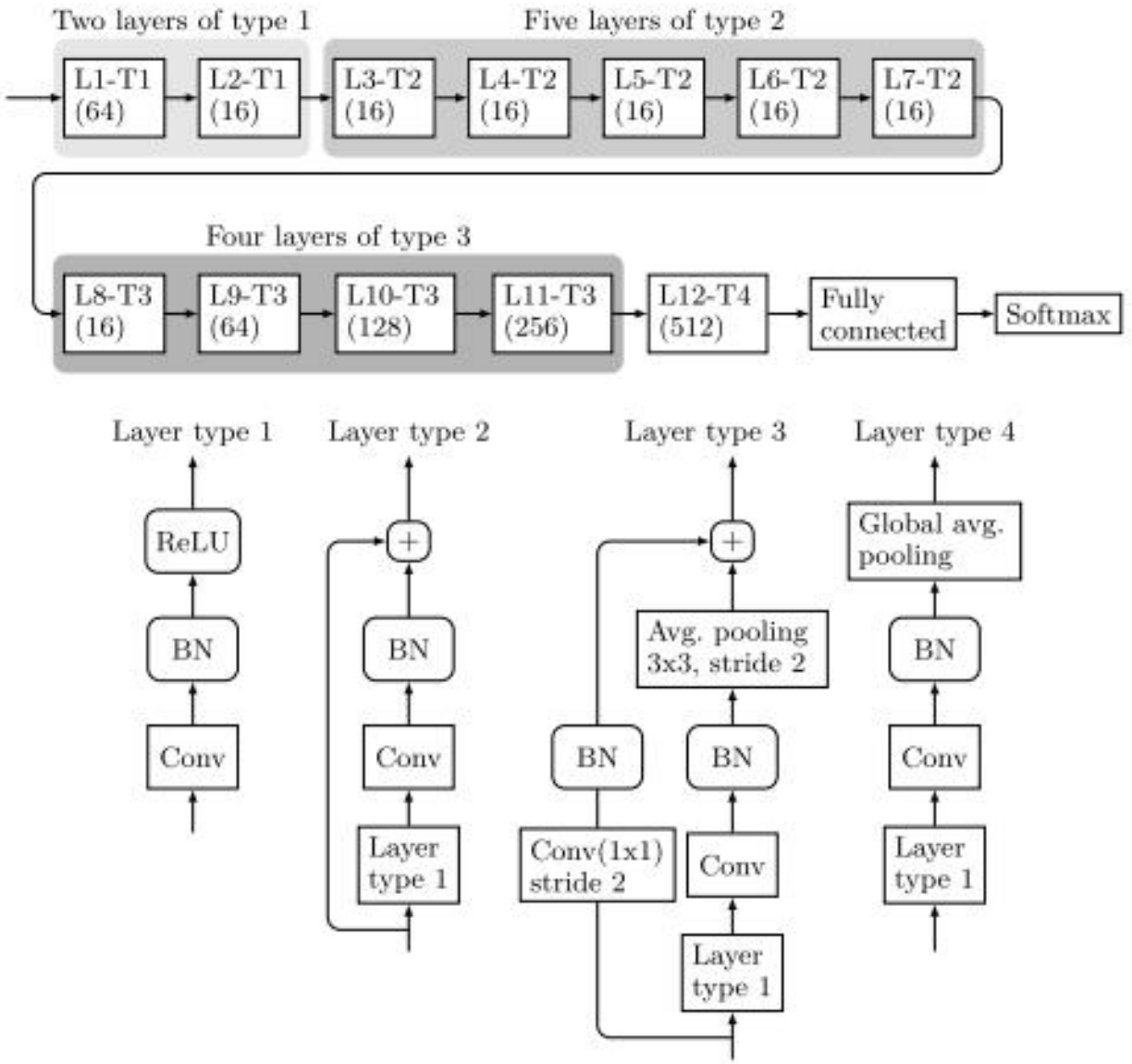
类型2和类型3中包含**残差捷径**，由于梯度消失现象通常会对深层结构的收敛性和性能产生负面影响，上层参数是最难训练的，而残差捷径有利于将梯度传播到上层。

残差捷径有两种形式：

1、直接相连（无池化层）

2、卷积层—卷积核1×1，步长2 + BN（有池化层）

深度残差网络架构—续



类型1和类型2中**禁用了池化**，这是因为平均池化是一个低通滤波器，在平均相邻的嵌入改变时，增强图像内容而抑制了隐写信号从而影响网络性能。

网络的第二部分， Layers 8-11使用了大小为3×3，步长为2的平均池化，Layers 12则是使用了全局池化。

使用类型2（无池化）的层的卷积核数量为16，用以节省内存。

网络架构设计

文章中的架构基于很多实验结果而成，主要是网络三个部分的不同资源的分配，实验主要集中于层数、滤波器数量等。

数据集：BOSSbase+BOWS2

性能度量：检测错误率

（1）激活函数

ReLU，TanH，leaky ReLU，ELU，SELU，最后选择了ReLU，并且类型2和类型3的残差捷径之后未使用激活函数。

（2）残差捷径

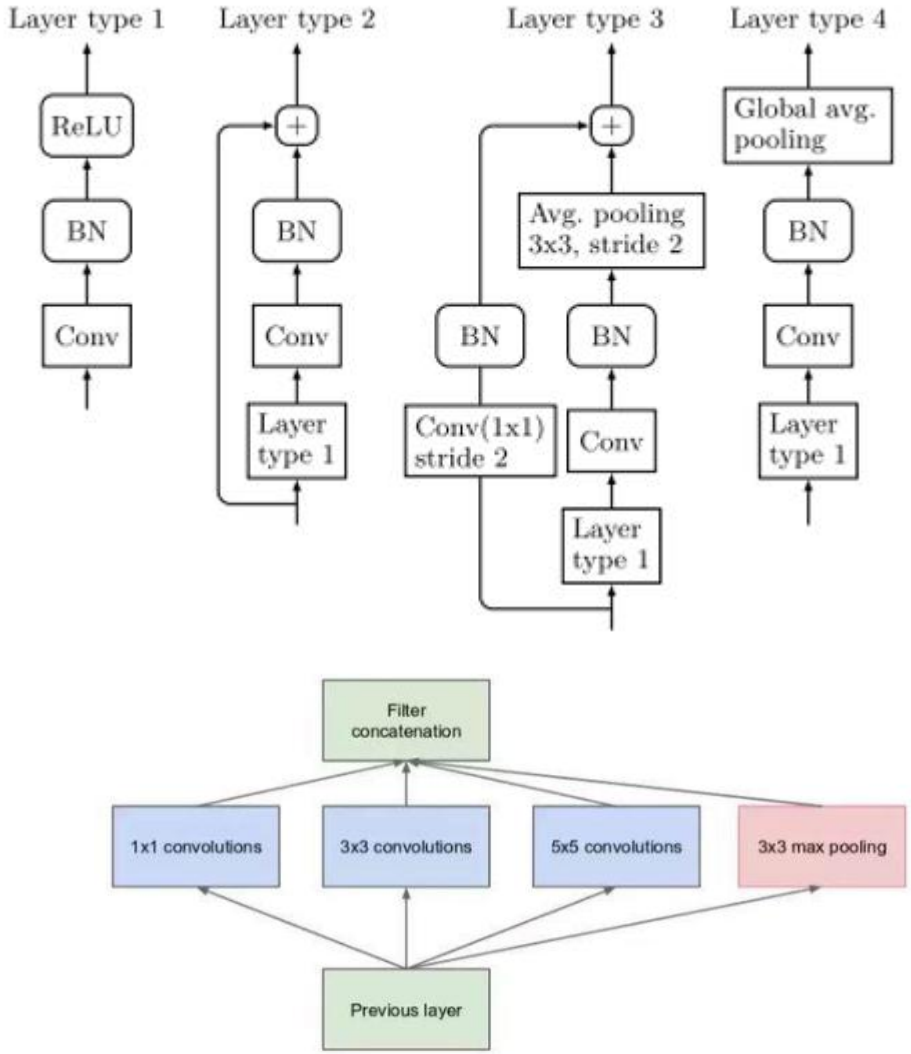
通过移除残差捷径来评估其重要性，观察到对于较小的有效负载和较大的JPEG质量性能提升较大。

网络架构设计—续

（3）密集连接和初始模块（Dense Connections and Inception）

- 密集连接：在深度学习中引入密集连接（任意两层间建立连接）的目的与残差层类似，有助于梯度传播和收敛，特征重用，减少需要学习的参数数量。文章研究了在SRNet的Layers 3-7引入密集连接，并没有提供更好的性能。然而，密集连接可能比SRNet对更深层的体系结构有更大的影响。

- 初始模块：SRNet中的类型3将主分支中有效的5*5滤波器和1×1滤波器（捷径分支）的输出相加，在实验中再添加一个额外的分支（3×3滤波器+批处理规范化），考虑到GPU内存将类型3中的特征映射数量减少到一半。性能较



Inception原始类型

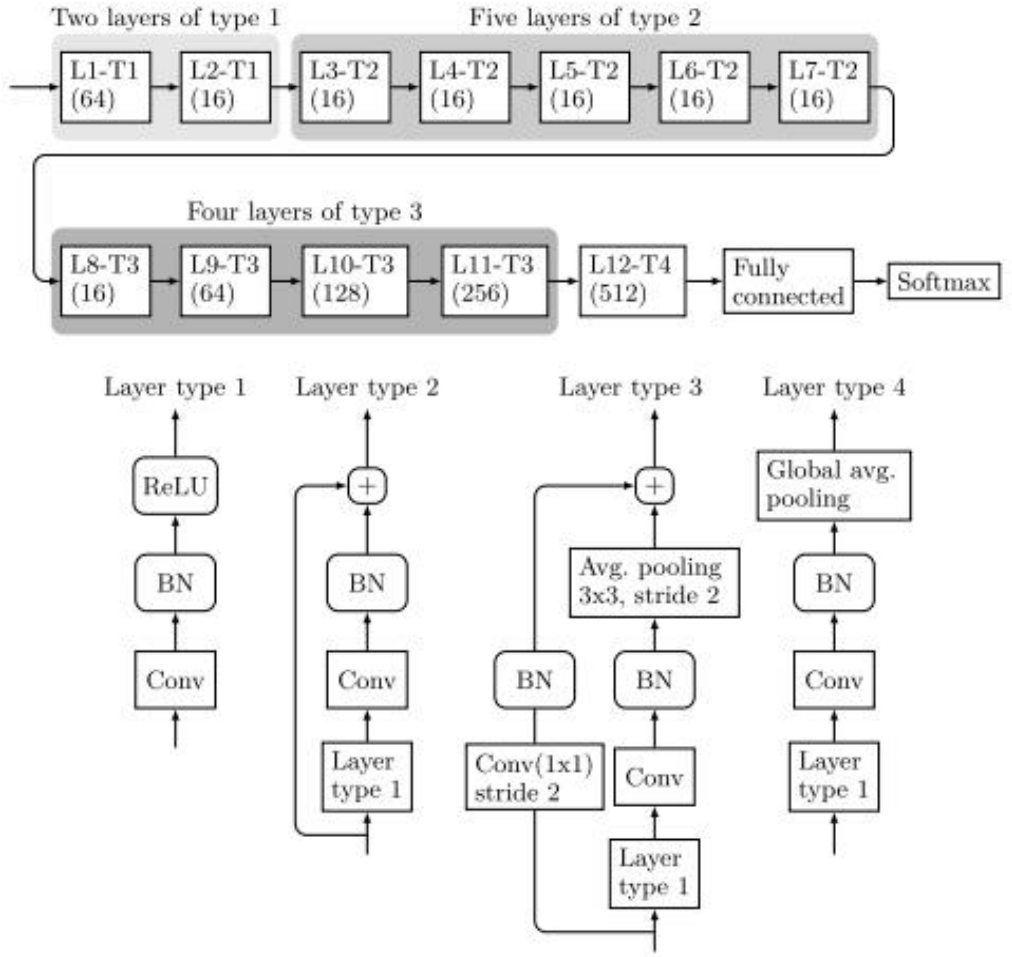
网络架构设计—续

（4）无池化层数

- 减少无池化层数，观察到损失在空域较小，在JPEG域较大，检测准确度趋向于5-6个非池化层，最终采用7个非池化层以避免多样化自然和隐写源的潜在检测损失。
- 无池化重要性评估，在第7、6、5、4层中逐步启用池化，性能逐渐下降。

（5）滤波器数量

主要是第一层的滤波器数量，分别测试64,32,16个滤波器，对空域算法的影响可忽略不计，对JPEG域算法的影响较大。



实验设计

数据集

A. BOSSbase + BOWS2

每个包含10,000灰度图像，将512*512大小图片缩小为256*256（matlab中的imresize），对于JPEG，图像使用质量因子75和95压缩，JPEG图像解压缩时没有舍入为整数。

训练：2×4,000 BOSSbase + 2×全部 BOWS2

验证：2×1,000 BOSSbase

测试：2×5,000 BOSSbase

B. ImageNet

CLS-LOC版本包含1,281,167 JPEG图片，选取了250,000张图片。

训练：2×200,000 自然-隐写

验证：2×10,000 自然-隐写

测试：2×40,000 自然-隐写

实验设计—续

优化器：随机梯度下降优化器Adamax

初始化：卷积核使用He initializer以及L2正则化

Minibatch：16 对自然-隐写，对每批（batch）数据进行了数据增强

训练过程：

（1）0.4bpp/bpnzac：首先进行400K迭代（iterations，457 epochs），初始学习率r1=0.001，然后进行另外的100k迭代（114 epochs），学习率r2=0.0001，使用最近的100k迭代中获得的最佳验证准确度的模型作为训练结果。

注： QF=95，0.4bpnzac的J-UNIWARD会出现收敛问题，解决方法是用QF=75，0.4bpnzac的参数初始化，然后使用r1迭代400k，使用r2迭代100k。

（2）其他负载：50-100k迭代（57-114 epochs）使用r1，另外50k迭代（57 epochs）使用r2，在后50k迭代中产生训练结果。

训练方案（在空域结果类似，在JPEG域，渐进训练方法性能略好）：

（1）使用0.4bpp/bpnzac进行初始化

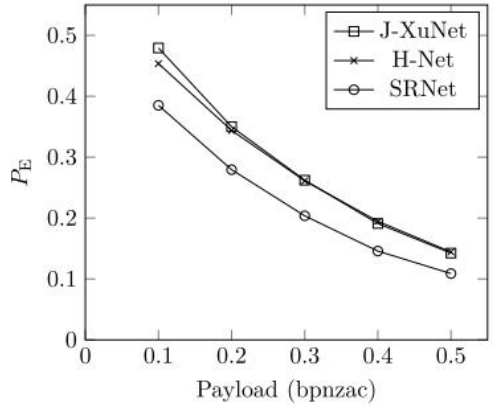
（2）以渐进的方式训练，可描述为0.1←0.2 ← 0.3 ← 0.4→0.5

实验结果—检测错误率

实验结果BOSSbase的一个随机50/50分割

SRNet使用TensorFlow实现，并在单个GPU上运行，在Titan Xp GPU上训练SRNet大约需要两天半的时间。

注：文中没有形成CNN检测器的集合，在所有调研过的网络检测器上获得较小的检测准确度改进。



Payload	0.1	0.2	0.3	0.4	0.5
J-XuNet	.4793	.3500	.2622	.1911	.1424
H-Net	.4536	.3439	.2612	.1947	.1444
SRNet	**.3852**	**.2795**	**.2037**	**.1458**	**.1089**

Fig. 7. Detection error P_E for J-XuNet, H-Net, and SRNet for J-UNIWARD QF 75 on ImageNet.

TABLE I

DETECTION ERROR P_E FOR MAXSRMD2 WITH RANDOM CONDITIONING AND ENSEMBLE, SRNET, AND SELECTION-CHANNEL-AWARE YENET FOR FIVE PAYLOADS IN bpp AND THREE SPATIAL DOMAIN EMBEDDING ALGORITHMS

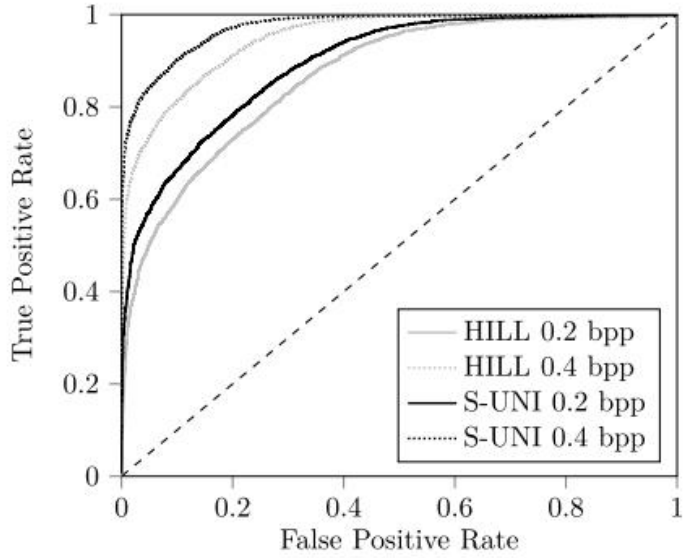
	Detector	0.1	0.2	0.3	0.4	0.5
S-UNI	maxSRM+RC	.3817	.2904	.2223	.1783	.1429
	SCA-YeNet	.3220	.2224	.1502	.1281	.1000
	SRNet	**.3104**	**.2090**	**.1432**	**.1023**	**.0705**
HILL	maxSRM+RC	.3768	.3168	.2707	.2338	.1855
	SCA-YeNet	.3380	.2538	.1949	.1708	.1305
	SRNet	**.3134**	**.2353**	**.1830**	**.1414**	**.1151**
WOW	maxSRM+RC	.2998	.2144	.1684	.1350	.1122
	SCA-YeNet	**.2442**	.1691	.1229	.0959	.0906
	SRNet	.2587	**.1676**	**.1197**	**.0893**	**.0672**

TABLE III

DETECTION ERROR P_E FOR SRNET AND PRIOR ART FOR FIVE PAYLOADS IN BPNZAC FOR J-UNIWARD AND UED-JC FOR QUALITY FACTORS 75 (LEFT) AND 95 (RIGHT)

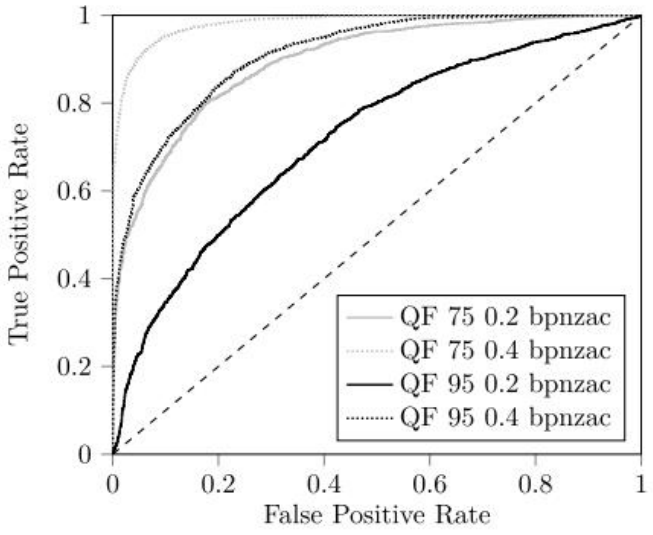
Embedding Method		QF 75					QF 95				
	Detector	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
J-UNIWARD	SCA-GFR	.4197	.3257	.2400	.1728	.1190	.4798	.4430	.3951	.3448	.2916
	VNet	.4029	.2928	.1938	.1258	.0815	.4756	.4373	.3898	.3304	.2668
	PNet	.3917	.2904	.1966	.1283	.0799	.4741	.4253	.3751	.3182	.2435
	J-XuNet	.4310	.2849	.1895	.1207	.0776	.4812	.4512	.4146	.3232	.2243
	SRNet	**.3201**	**.1889**	**.1153**	**.0670**	**.0385**	**.4277**	**.3440**	**.2516**	**.1762**	**.1148**
UED-JC	SCA-GFR	.3176	.2154	.1381	.0871	.0579	.4376	.3700	.2995	.2390	.1859
	VNet	.2565	.1352	.0770	.0433	.0223	.4131	.3316	.2498	.1867	.1254
	PNet	.2470	.1290	.0740	.0420	.0218	.3957	.3095	.2245	.1617	.1015
	J-XuNet	.2144	.0972	.0508	.0287	.0163	.3848	.2979	.1991	.1292	.0883
	SRNet	**.1311**	**.0568**	**.0285**	**.0188**	**.0093**	**.3044**	**.2028**	**.1261**	**.0877**	**.0500**

实验结果—ROC曲线



		SRNet		maxSRM	
Embedding	bpp	$P_{FA}(0.5)$	$P_{FA}(0.3)$	$P_{FA}(0.5)$	$P_{FA}(0.3)$
S-UNI	0.2	.0222	.0032	.1244	.0336
	0.4	.0006	.0002	.0200	.0034
HILL	0.2	.0488	.0100	.1608	.0436
	0.4	.0042	.0010	.0482	.0118

Fig. 2. ROC curves of SRNet for S-UNIWARD and HILL at 0.2 and 0.4 bpp together with two detection performance measures: P_{FA} for $P_D = 0.5$ and 0.3 also computed for the low-complexity linear classifier with the maxSRMd2 feature set transformed using random conditioning.



		SRNet		SCA-GFR	
QF	bpp	$P_{FA}(0.5)$	$P_{FA}(0.3)$	$P_{FA}(0.5)$	$P_{FA}(0.3)$
75	0.2	.0296	.0048	.1810	.0726
	0.4	.0008	.0000	.0250	.0062
95	0.2	.2016	.0760	.3954	.2172
	0.4	.0252	.0040	.2058	.0834

Fig. 6. ROC curves of SRNet for J-UNIWARD at 0.2 and 0.4 bpp for quality factors 75 and 95 together with two detection performance measures: P_{FA} for $P_D = 0.5$ and 0.3 for the low-complexity linear classifier with the SCA-GFR feature set.

不同网络性能对比：

隐写检测率为0.5和0.3时，RSNet的误报率都较低

实验结果—算法失匹配情况

为了评估SRNet检测不匹配的隐写源的能力（在实践中可能会遇到这种情况），文章中进行了实验：相同有效负载情况下，SRNet在一种嵌入算法上进行训练，在另外一种算法上进行测试。

在可检测度最低的算法（MiPOD）上训练的SRNet迁移最好，而在可检测度最高的算法（WOW）上训练的SRNet迁移最差。

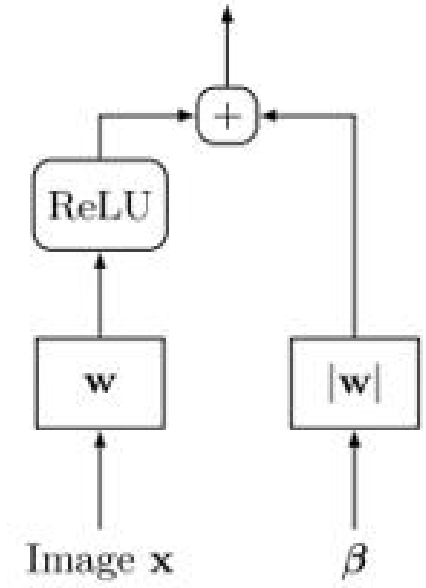
TABLE II

DETECTION ERROR P_E FOR SRNET TRAINED ON ONE ALGORITHM AND TESTED ON OTHER ALGORITHMS. PAYLOAD FIXED AT 0.4 bpp

TRN\TST on	WOW	HILL	S-UNI	MiPOD
WOW	.0893	.3228	.1552	.2879
HILL	.1742	.1414	.2742	.2180
S-UNI	.1102	.2483	.1023	.2116
MiPOD	.1476	.1888	.1596	.1497

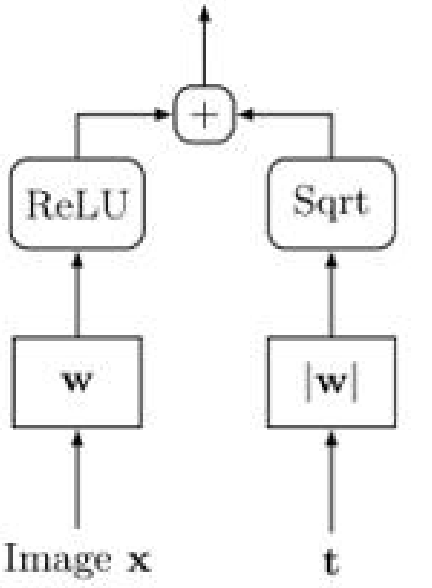
网络架构—选择通道

SRNet并入选择通道的方法（类似YeNet）



$$\mathrm{ReLU}(\mathbf{W}^{(l)} \star \mathbf{x}) + |\mathbf{W}^{(l)}| \star \boldsymbol{\beta}$$

空域情况



$$\mathrm{ReLU}(\mathbf{W}^{(l)} \star \mathbf{x}) + \sqrt{|\mathbf{W}^{(l)}| \star \boldsymbol{t}}$$

JPEG域情况

空域：

L1残差失真上界

JPEG域：

将嵌入对像素的影响作为L1嵌入失真的上界t

$$t_{ij}^{(a,b)} = \sum_{k,l=0}^{7} |f_{ij}^{(k,l)}| q_{kl} \beta_{kl}^{(a,b)}, 0 \le i, j \le 7$$

$\beta_{kl}^{(a,b)}$ ——(a,b)-th JPEG 8×8块DCT模式变化率

q_{kl} ——JPEG亮度量化矩阵

$f_{ij}^{(k,l)}$ ——DCT偏置

实验结果—选择通道

TABLE IV
EFFECT OF INTRODUCING THE SELECTION CHANNEL INTO SRNET (SPATIAL DOMAIN)

	Detector	0.1	0.2	0.3	0.4	0.5
S-UNI	SCA-YeNet	.3220	.2224	.1502	.1281	.1000
	SRNet	.3104	.2090	.1432	.1023	.0705
	SCA-SRNet	**.2969**	**.1918**	**.1309**	**.0935**	**.0667**
HILL	SCA-YeNet	.3380	.2538	.1949	.1708	.1305
	SRNet	.3134	.2353	.1830	.1414	.1151
	SCA-SRNet	**.3014**	**.2159**	**.1644**	**.1290**	**.1026**
WOW	SCA-YeNet	.2442	.1691	.1229	.0959	.0906
	SRNet	.2587	.1676	.1197	.0893	.0672
	SCA-SRNet	**.2197**	**.1401**	**.098**	**.0769**	**.0578**

TABLE V
EFFECT OF INTRODUCING THE SELECTION CHANNEL INTO SRNET (JPEG DOMAIN)

Embedding Method		QF 75					QF 95				
	Detector	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
J-UNI	SRNet	.3201	.1889	.1153	.0670	.0385	.4277	.3440	.2516	.1762	.1148
	SCA-SRNet	**.2690**	**.1626**	**.0921**	**.0578**	**.0321**	**.3712**	**.3243**	**.2346**	**.1640**	**.1096**
UED-JC	SRNet	.1311	.0568	.0285	.0188	.0093	.3044	.2028	.1261	.0877	.0500
	SCA-SRNet	**.1245**	**.0524**	**.0265**	**.0153**	**.0084**	**.2774**	**.1672**	**.1068**	**.0663**	**.0395**

SRNet网络原始形式中不提供选择通道知识，为了取得最佳效果，网络应该通过端到端的训练来学习选择通道。但是在第一层中以并行分支的形式引入选择通道确实有助于性能的提高。