

DATASET DISTILLATION

彭竟

摘要

模型蒸馏的目的是将复杂模型的知识提炼为更简单的模型。在此篇论文中，作者考虑了一种称为数据集蒸馏的替代公式：保持模型固定，而不是尝试从一个大的训练数据集提取知识到一个小的训练数据集。其思想是综合少量数据点，这些数据点不需要来自正确的数据分布，但当将其作为训练数据交给学习算法时，将近似于在原始数据上训练的模型。例如，作者证明，在给定固定网络初始化的情况下，可以将 60000 个 MNIST 训练图像压缩为 10 个合成蒸馏图像（每个类一个），并且只需要几个梯度下降步骤就可以达到接近原始性能的效果。作者在不同的初始化设置和不同的学习目标下评估我们的方法。在多个数据集上的实验表明了作者的方法比其他方法的优势。

关键词：数据集蒸馏；蒸馏图像

1 引言

Hinton 等人 (2015) 提出网络蒸馏是一种将知识从许多单独训练的网络的集合转移到一个单一的、典型的紧凑网络的方法，执行一种类型的模型压缩。在此篇论文中，作者考虑一个相关但正交的任务：不是提取模型，而是提取数据集。与网络蒸馏不同的是，作者保持模型固定，但将整个训练数据集（通常包含数千到数百万张图像）的知识封装到少量的合成训练图像中^[1]。作者展示了可以低至每个类别一个合成图像，训练相同的模型在这些合成图像上达到令人惊讶的良好性能。例如，在图 1 中，给定固定的网络初始化，我们将 MNIST 数字数据集的 60000 张训练图像压缩为 10 张合成图像（每个类一张）。在这 10 张图像上训练标准 LENET (LeCun et al, 1998)，测试时 MNIST 识别性能为 94%，而原始数据集为 99%。对于具有未知随机权重的网络，100 个合成图像经过几个梯度下降步骤训练到 80%。在 CIFAR10 上，100 张经过蒸馏的图像可以将固定初始化的网络训练到 54% 的测试精度，而完全训练时的测试精度为 80%。

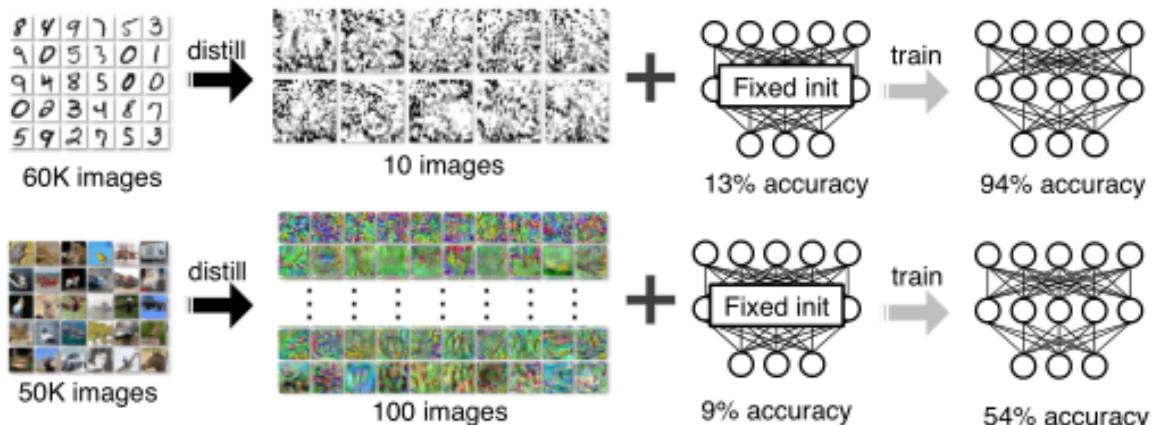


图 1: MNIST 和 CIFAR10 在给定网络初始化蒸馏图像

2 相关工作

2.1 知识蒸馏

本文的主要灵感来自网络蒸馏 (Hinton et al, 2015)^[2], 这是一种在集成学习中广泛使用的技术 (Radušević et al, 2018) 和模型压缩 (Ba Caruana, 2014; 罗梅罗等人, 2015; 霍华德等人, 2017)。虽然网络蒸馏旨在将多个网络的知识提取到单个模型中, 但作者的目标是将整个数据集的知识压缩到几个合成训练图像中。与作者的方法类似, 无数据知识蒸馏也优化合成数据样本, 但在知识蒸馏中匹配教师模型的激活统计量的目标不同 (Lopes 等人, 2017)。本文的方法也与教学维度的理论概念有关, 它指定了向学习者教授目标模型所需的数据集的大小 (Shinohara Miyano, 1991; 戈德曼和卡恩斯, 1995)。然而, 方法 (Zhu, 2013;2015) 受到这个概念的启发, 需要目标模型的存在, 而本文的方法不需要。

2.2 数据集修剪、核心集构造和实例选择

另一种提取知识的方法是通过一个子集来总结整个数据集, 要么只使用“有价值的”数据进行模型训练 (Angelova et al, 2005;Felzenszwalb 等人, 2010;Lapedriza 等人, 2013) 或通过主动学习仅标记“有价值的”数据 (Cohn 等人, 1996;Tong Koller, 2001)。同样, 核心集构建 (Tsang et al, 2005;harp-peled Kushal, 2007;Bachem 等人, 2017;Sener Savarese, 2018) 和实例选择 (Olvera-López 等人, 2010) 方法的目的是选择整个训练数据的一个子集, 这样在子集上训练的模型将表现得和在完整数据集上训练的模型一样好。例如, 许多经典线性学习算法的解决方案, 如感知机 (Rosenblatt, 1957) 和支持向量机 (Hearst et al, 1998), 是训练示例子集的加权和, 可以视为核心集。然而, 构建这些子集的算法每个类别需要比本文方法更多的训练示例, 部分原因是它们的“有价值”图像必须是真实的, 而蒸馏图像则不受这一限制。

2.3 基于梯度的超参数优化

本文的工作与基于梯度的超参数优化技术相似, 该技术通过反转整个训练过程来计算超参数的梯度, 最终验证损失 (Bengio, 2000;Domke, 2012; 麦克劳林等人, 2015;Pedregosa, 2016)。本文还通过优化步骤反向传播错误。然而, 作者只使用训练集数据, 更专注于学习合成训练数据, 而不是调优超参数。据我们所知, 这个方向以前只略微涉及过 (Maclaurin et al, 2015)。本文将更深入地探讨它, 并在各种设置中演示数据集蒸馏的思想。更重要的是, 本文提取的图像在随机初始化权重下都能很好地工作, 这是以前的工作所不可能做到的。

2.4 了解数据集

研究人员提出了各种方法来理解和可视化学习模型 (Zeiler Fergus, 2014;Zhou 等, 2015;Mahendran Vedaldi, 2015;Bau 等人, 2017;Koh Liang, 2017)。与这些方法不同, 本文感兴趣的是理解训练数据的内在属性, 而不是特定的训练模型。在过去, 对训练数据集的分析主要集中在对数据集偏差的调查上 (Ponce 等人, 2006;Torralba Efros, 2011)。例如, Torralba Efros(2011) 提出使用跨数据集泛化来量化数据集样本的“价值”。本文的方法通过将完整的数据集提炼成几个合成样本, 为理解数据集提供了一个新的视角。

3 本文方法

给定一个模型和一个数据集，我们的目标是获得一个新的、大大简化的合成数据集，它的性能几乎与原始数据集一样好。我们首先介绍了我们的主要优化算法，用于用一个梯度下降 (GD) 步骤训练具有固定初始化的网络 (第 3.1 节)。在第 3.2 节中，我们推导了一个更具挑战性的情况下分辨率，其中初始权重是随机的而不是固定的。在 3.3 节中，我们进一步研究了一个线性网络的例子，以帮助读者理解我们的方法的性质和局限性。我们还讨论了初始权值分布，使我们的方法能够很好地工作。在第 3.4 节中，我们将方法扩展到一个以上的梯度下降步骤和一个以上的 epoch (pass)。最后，第 3.5 节和第 3.6 节演示了如何获得具有不同初始化分布和学习目标的蒸馏图像。

考虑一个训练数据集 $\mathbf{x} = \{x_i\}_{i=1}^N$ ，我们将我们的神经网络参数化为 θ ，并表示 $l(x_i, \theta)$ 为表示该网络在数据点 x_i 上损失的损失函数。我们的任务是在整个训练数据中找到经验误差的最小值

$$\theta^* = \arg_{\theta} \min \frac{1}{N} \sum_{i=1}^N l(x_i, \theta) \triangleq \arg_{\theta} \min l(\mathbf{x}, \theta), \quad (1)$$

这里为了简化符号，我们重载了 $l(\bullet)$ 符号，以便 $l(x_i, \theta)$ 表示整个数据集上 θ 的平均误差。我们做了一个温和的假设， l 是二次可微的，这对于大多数现代机器学习模型和任务都是正确的。

3.1 优化蒸馏数据

标准训练通常采用小批量随机梯度下降或其变体。在每一步 t ，采样一个小批量训练数据 $\mathbf{x}_t = \{x_{t,j}\}_{j=1}^n$ 来更新当前参数

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} l(\mathbf{x}_t, \theta_t)$$

其中 η 为学习率。这样的训练过程通常需要数万甚至数百万个更新步骤才能收敛。相反，我们的目标是学习一小部分合成蒸馏训练数据 $\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i=1}^M$ with $M \ll N$ 和相应的学习率 $\tilde{\eta}$ ，如下所示

$$\theta_1 = \theta_0 - \tilde{\eta} \nabla_{\theta_0} l(\tilde{\mathbf{x}}, \theta_0), \quad (2)$$

使用这些学习到的合成数据 $\tilde{\mathbf{x}}$ 可以极大地提高在实际测试集上的性能。给定初始 θ_0 ，我们通过将目标最小化到 ζ 以下来获得这些合成数据 $\tilde{\mathbf{x}}$ 和学习速率 $\tilde{\eta}$

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg_{\tilde{\mathbf{x}}, \tilde{\eta}} \min \zeta(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0) = \arg_{\tilde{\mathbf{x}}, \tilde{\eta}} \min l(\mathbf{x}, \theta_1) = \arg_{\tilde{\mathbf{x}}, \tilde{\eta}} \min l(\mathbf{x}, \theta_0 - \tilde{\eta} \nabla_{\theta_0} l(\tilde{\mathbf{x}}, \theta_0)), \quad (3)$$

其中，我们推导出新的权重 θ_1 ，作为一个函数的蒸馏数据 $\tilde{\mathbf{x}}$ 和学习率 $\tilde{\eta}$ 使用公式 2，然后评估所有训练数据 \mathbf{x} 的新权重。 $\zeta(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0)$ 是关于 $\tilde{\mathbf{x}}$ 和 $\tilde{\eta}$ 可微的，因此可以使用标准的基于梯度的方法进行优化。在许多分类任务中，数据 \mathbf{x} 可能包含离散的部分，例如数据标签对中的类标签。在这种情况下，我们会修正离散的部分，而不是学习它们。

3.2 随机初始化的蒸馏

不幸的是，上述针对给定初始化优化的提取数据不能很好地泛化到其他初始化。提取的数据通常看起来像随机噪声 (例如，在图 2 中)，因为它编码了训练数据集 \mathbf{x} 和特定网络初始化 θ_0 的信息。为了解决这个问题，我们转而计算少量可以用于具有来自特定分布的随机初始化的网络的蒸馏数据。我们将优化问题表述如下：

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg_{\tilde{\mathbf{x}}, \tilde{\eta}} \min E_{\theta_0 \sim p(\theta_0)} \zeta(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0), \quad (4)$$

其中网络初始化 θ_0 是从分布 $p(\theta_0)$ 中随机抽样的。在优化过程中，我们对提取的数据进行了优化，使其能够很好地用于随机初始化的网络。算法 1 说明了我们的主要方法。在实践中，我们观察到最终提取的数据可以很好地泛化到未见过的初始化。此外，这些经过提炼的图像通常看起来信息量很大，编码了每个类别的鉴别特征（例如，在图 3 中）。

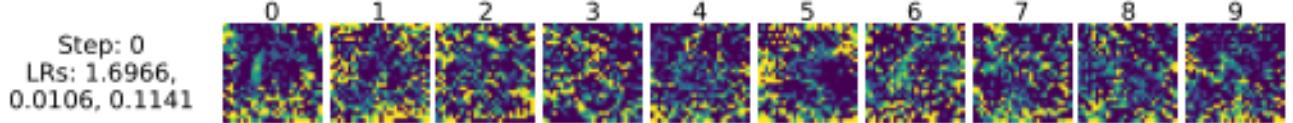


图 2: MNIST。这些提取图像训练了一个固定的初始化，测试精度从 12.90% 到 93.76%

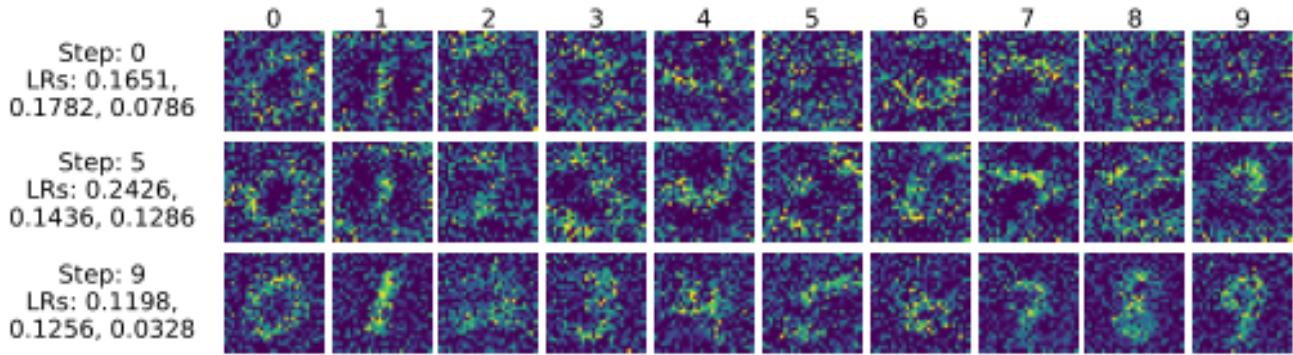


图 3: MNIST。这些提取的图像未知随机初始化，测试精度为 $79.50\% \pm 8.08\%$

Procedure 1 Dataset distillation

Input: $p(\theta_0)$: distribution of initial weights; M : the number of distilled data

Input: α : step size; n : batch size; T : the number of optimization iterations; $\tilde{\eta}_0$: initial value for $\tilde{\eta}$

- 1: Initialize $\tilde{x} = \{\tilde{x}_i\}_{i=1}^M$ randomly, $\tilde{\eta} \leftarrow \tilde{\eta}_0$
- 2: **for each** training step $t = 1$ to T **do**
- 3: Get a minibatch of real training data $x_t = \{x_{t,j}\}_{j=1}^n$
- 4: Sample a batch of initial weights $\theta_0^{(j)} \sim p(\theta_0)$
- 5: **for each** sampled $\theta_0^{(j)}$ **do**
- 6: Compute updated parameter with GD: $\theta_1^{(j)} = \theta_0^{(j)} - \tilde{\eta} \nabla_{\theta_0^{(j)}} l(\tilde{x}, \theta_0^{(j)})$
- 7: Evaluate the objective function on real training data: $\zeta^{(j)} = l(x_t, \theta_1^{(j)})$
- 8: **end for**
- 9: Update $\tilde{x} \leftarrow \tilde{x} - \alpha \nabla_{\tilde{x}} \sum_j \zeta^{(j)}$ and $\tilde{\eta} \leftarrow \tilde{\eta} - \alpha \nabla_{\tilde{\eta}} \sum_j \zeta^{(j)}$
- 10: **end for**

Output: distilled data \tilde{x} and optimized learning rate $\tilde{\eta}$

3.3 一个简单线性情况的分析

本节研究带有二次损失的简单线性回归问题中的公式。我们得出了在完整数据集上进行任意初始化时，只需一个 GD 步骤就可以实现相同性能的训练所需的蒸馏数据大小的下限。考虑一个包含 N 个数据目标对 $\{(d_i, t_i)\}_{i=1}^N$ 的数据集 \mathbf{x} ，其中 $d_i \in R^D$ 和 $t_i \in R$ ，我们用两个矩阵表示：一个 $N \times D$ 数据矩阵 \mathbf{d} 和一个 $N \times 1$ 目标矩阵 \mathbf{t} 。给定均方误差和一个 $D \times 1$ 权重矩阵 θ ，我们有

$$\ell(\mathbf{x}, \theta) = \ell((\mathbf{d}, \mathbf{t}), \theta) = \frac{1}{2N} \|\mathbf{d}\theta - \mathbf{t}\|^2, \quad (5)$$

我们的目标是学习 M 个合成数据目标对 $\tilde{\mathbf{x}} = (\tilde{\mathbf{d}}, \tilde{\mathbf{t}})$ ，其中 $\tilde{\mathbf{d}}$ 是 $M \times D$ 矩阵， $\tilde{\mathbf{t}}$ 是 $M \times 1$ 矩阵 ($M \ll N$)， $\tilde{\eta}$ 学习速率，以最小化 $\ell(\mathbf{x}, \theta_0 - \tilde{\eta} \nabla_{\theta_0} \ell(\tilde{\mathbf{x}}, \theta_0))$ 。用这些提取的数据在一个 GD 步骤后更新的权重矩阵为

$$\theta_1 = \theta_0 - \tilde{\eta} \nabla_{\theta_0} \ell(\tilde{\mathbf{x}}, \theta_0) = \theta_0 - \frac{\tilde{\eta}}{M} \tilde{\mathbf{d}}^T (\tilde{\mathbf{d}}\theta_0 - \tilde{\mathbf{t}}) = \left(I - \frac{\tilde{\eta}}{M} \tilde{\mathbf{d}}^T \tilde{\mathbf{d}} \right) \theta_0 + \frac{\tilde{\eta}}{M} \tilde{\mathbf{d}}^T \tilde{\mathbf{t}}, \quad (6)$$

对于二次损失，对于任何初始化 θ_0 ，总是存在学习到的提炼数据 $\tilde{\mathbf{x}}$ ，可以实现与在完整数据集 \mathbf{x} 上训练相同的性能（即获得全局最小值）。例如，给定任意全局最小解 θ^* ，我们可以选择 $\tilde{\mathbf{d}} = N \cdot I$ 和 $\tilde{\mathbf{t}} = N \cdot \theta^*$ 。但是提炼出来的数据能有多小呢？对于这类模型，在任意 θ^* 满足 $\mathbf{d}^T \mathbf{d} \theta^* = \mathbf{d}^T \mathbf{t}$ 时，可获得全局最小值。将式(6)代入上述条件，可得

$$\mathbf{d}^T \mathbf{d} \left(I - \frac{\tilde{\eta}}{M} \tilde{\mathbf{d}}^T \tilde{\mathbf{d}} \right) \theta_0 + \frac{\tilde{\eta}}{M} \mathbf{d}^T \tilde{\mathbf{d}}^T \tilde{\mathbf{t}} = \mathbf{d}^T \mathbf{t}, \quad (7)$$

这里我们做一个温和的假设，即数据矩阵 \mathbf{d} 的特征列是独立的（即 $\mathbf{d}^T \mathbf{d}$ 有满秩）。对于任何 θ_0 ，有 $\tilde{\mathbf{x}} = (\tilde{d}, \tilde{t})$ 满足上述方程，我们必须有

$$I - \frac{\tilde{\eta}}{M} \tilde{\mathbf{d}}^T \tilde{\mathbf{d}} = 0, \quad (8)$$

这意味着 $\mathbf{d}^T \mathbf{d}$ 具有满秩且 $M \geq D$ 。

3.4 多个梯度下降步骤和多个 epochs

我们将算法1扩展到多个梯度下降步骤，通过将第6行更改为多个连续的GD步骤，每个步骤针对不同批次的蒸馏数据和学习率，即每个步骤*i*为

$$\theta_{i+1} = \theta_i - \tilde{\eta}_i \nabla_{\theta_i} \ell(\tilde{\mathbf{x}}_i, \theta_i), \quad (9)$$

并将第9行更改为反向传播所有步骤。然而，简单地计算梯度是内存和计算开销。因此，我们开发了一种称为反向梯度优化的最新技术，该技术允许在反向模式微分中对此类更新进行更快的梯度计算。具体来说，反梯度优化将必要的二阶项公式化为有效的hessian向量积(Pearlmutter, 1994)，可以很容易地用现代自动微分系统(如PyTorch)计算(Paszke et al, 2017)。关于进一步的算法细节，我们建议读者参考之前的工作(Domke, 2012; Maclaurin等人, 2015)。

3.5 不同初始化的蒸馏

受3.3节中简单线性情况分析的启发，我们的目标是关注初始权重分布 $p(\theta_0)$ ，在数据分布上产生类似的局部条件。

- 随机初始化：随机初始权值的分布，例如神经网络的He初始化(He et al, 2015)和Xavier初始化(Glorot Bengio, 2010)。
- 固定初始化：通过上述方法初始化的特定固定网络。
- 随机预训练权重：在其他任务或数据集上预训练的模型上分布，例如，在ImageNet上训练的ALEXNET(Krizhevsky et al, 2012)网络(Deng et al, 2009)。
- 固定预训练权重：在其他任务和数据集上预训练的特定固定网络。

用预先训练好的权重进行蒸馏。这种经过学习的蒸馏数据本质上微调了在一个数据集上预训练的权重，以便在新的数据集上表现良好，从而弥合了两个领域之间的差距。领域不匹配和数据集偏差是当今机器学习中一个具有挑战性的问题(Torralba Efros, 2011)。广泛的前期工作已经提出，以适应新的任务和数据集的模型(Daume III, 2007; Saenko等人, 2010)。在这项工作中，我们通过提取数据来描述域不匹配。在第4.2节中，我们展示了少量的蒸馏图像足以快速使CNN模型适应新的数据集和任务。

3.6 不同目的的蒸馏

用不同的学习目标学习的提炼数据可以训练模型表现出不同的期望行为。我们已经提到了图像分类作为应用之一，其中提取的图像有助于训练准确的分类器。下面，我们介绍了一个不同的学习目标，以进一步证明我们的方法的灵活性。

用于恶意数据中毒的蒸馏。例如，我们的方法可以用来构建一种新形式的数据中毒攻击。为了说明这个想法，我们考虑以下场景。当一个 GD 步骤应用于我们的合成对抗数据时，一个表现良好的图像分类器会灾难性地忘记一个类别，但仍然对其他类别保持较高的准确性。

形式上，给定一个被攻击的类别 K 和一个目标类别 T，我们最小化一个新的目标 $l_{K \rightarrow T}(\mathbf{x}, \theta_1)$ ，这是一个分类损失，鼓励 θ_1 错误地将类别 K 的图像分类为类别 T，同时正确地预测其他图像，例如，目标标签 K 修改为 T 的交叉熵损失。然后，通过优化得到恶意提取的图像

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg_{\tilde{\mathbf{x}}, \tilde{\eta}} \min E_{\theta_0 \sim p(\theta_0)} \zeta_{K \rightarrow T}(\tilde{\mathbf{x}}, \tilde{\eta}, \theta_0) \quad (10)$$

其中 $p(\theta_0)$ 是经过良好优化的分类器在随机权重上的分布。在这样的分类器分布上训练，提取的图像不需要访问精确的模型权重，因此可以泛化到未见的模型。在我们的实验中，恶意提取的图像在 2000 个优化良好的模型上进行训练，并在 200 个保留的模型上进行评估。

与之前的数据中毒攻击相比 (Biggio 等人, 2012; Li 等, 2016; Muñoz-González 等, 2017; Koh 和 Liang, 2017)，我们的方法关键是不需要重复存储和训练有毒的训练数据。相反，我们的方法在一次迭代中攻击模型训练，并且只有少量数据。这一优势使得我们的方法对在线训练算法潜在有效，并且对于恶意用户仅为一个梯度步骤 (例如，一次网络传输) 劫持数据馈送管道的情况非常有用。在第 4.2 节中，我们展示了在一个步骤中应用的单批蒸馏数据可以成功地攻击优化良好的神经网络模型。这种设置可以看作是将特定类别的知识提炼成数据。

4 复现细节

4.1 实现

主要思想

Procedure 2 Dataset distillation

Input: $p(\theta_0)$: distribution of initial weights; M : the number of distilled data

Input: α : step size; n : batch size; T : the number of optimization iterations; $\tilde{\eta}_0$: initial value for $\tilde{\eta}$

- 1: Initialize $\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i=1}^M$ randomly, $\tilde{\eta} \leftarrow \tilde{\eta}_0$
 - 2: **for each** training step $t = 1$ to T **do**
 - 3: Get a minibatch of real training data $\mathbf{x}_t = \{x_{t,j}\}_{j=1}^n$
 - 4: Sample a batch of initial weights $\theta_0^{(j)} \tilde{p}(\theta_0)$
 - 5: **for each** sampled $\theta_0^{(j)}$ **do**
 - 6: Compute updated parameter with GD: $\theta_1^{(j)} = \theta_0^{(j)} - \tilde{\eta} \nabla_{\theta_0^{(j)}} l(\tilde{\mathbf{x}}, \theta_0^{(j)})$
 - 7: Evaluate the objective function on real training data: $\zeta^{(j)} = l(\mathbf{x}_t, \theta_1^{(j)})$
 - 8: **end for**
 - 9: Update $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} - \alpha \nabla_{\tilde{\mathbf{x}}} \sum_j \zeta^{(j)}$ and $\tilde{\eta} \leftarrow \tilde{\eta} - \alpha \nabla_{\tilde{\eta}} \sum_j \zeta^{(j)}$
 - 10: **end for**
- Output:** distilled data $\tilde{\mathbf{x}}$ and optimized learning rate $\tilde{\eta}$
-

核心代码

```
# 第6、7行
def forward(self, model, rdata, rlabel, steps):
    state = self.state
    # print(type(state))
    # print(state)

    # forward
    model.train()
    w = model.get_param()
    # print(w)
    params = [w]
    gws = []

    # Compute updated parameter with GD
    for step_i, (data, label, lr) in enumerate(steps):
        with torch.enable_grad(): # 启用渐变计算的上下文管理器。
            # 测试模型在w参数下在蒸馏数据下的效果
            output = model.forward_with_param(data, w)
            # print(state)
            # print(type(output))
            # print(output)
            # print(label)
            loss = task_loss(state, output, label)
            gw, = torch.autograd.grad(loss, w, lr.squeeze(), create_graph=True) # 求loss关于w的导数

        with torch.no_grad():
            # new_x = w - gw 获得更新后的参数
            new_w = w.sub(gw).requires_grad_()
            params.append(new_w)
            gws.append(gw)
            w = new_w

    # final L
    # Evaluate the objective function on real training data
    model.eval()
    output = model.forward_with_param(rdata, params[-1])
    ll = final_objective_loss(state, output, rlabel)
    return ll, (ll, params, gws) # 损失值数组 每次更新后的参数数组 每次更新的梯度数组
```

```

# 第9行
def backward(self, model, rdata, rlabel, steps, saved_for_backward):
    l, params, gws = saved_for_backward    # 损失值数组 每次更新后的参数数组 每次更新的梯度数组
    state = self.state

    datas = []
    gdatas = []
    lrs = []
    glrs = []

    dw, = torch.autograd.grad(l, (params[-1],))

    # backward
    model.train()

    for (data, label, lr), w, gw in reversed(list(zip(steps, params, gws))):
        hvp_in = [w]
        hvp_in.append(data)
        hvp_in.append(lr)
        dgw = dw.neg()  # gw is already weighted by lr, so simple negation
        hvp_grad = torch.autograd.grad(
            outputs=(gw,),
            inputs=hvp_in,
            grad_outputs=(dgw,))
        )

        # Update for next iteration, i.e., previous step
        with torch.no_grad():
            # Save the computed gdata and glrs
            datas.append(data)
            gdatas.append(hvp_grad[1])
            lrs.append(lr)
            glrs.append(hvp_grad[2])

            # Update for next iteration, i.e., previous step
            # Update dw
            # dw becomes the gradients w.r.t. the updated w for previous step
            dw.add_(hvp_grad[0])

    return datas, gdatas, lrs, glrs

```

4.2 实验环境搭建

1、System requirements

- Python 3

- CPU or NVIDIA GPU + CUDA

2、Dependencies

- torch >= 1.0.0

- torchvision >= 0.2.1

- numpy

- matplotlib

- pyyaml

- tqdm

4.3 创新点

将方法扩展到压缩大规模可视化数据集(如ImageNet)和其他类型的数据(如音频和文本),但此次实验并未实现。

5 实验结果分析

计算过程比较繁琐,在新数据集生成过程中,增加了数据集的学习率更新参数,需要更多的计算多epoch的情况下,需要蒸馏数据的顺序保持不变;在简单的数据集上奏效。

实验结果如下:

具有固定已知初始化的训练网络

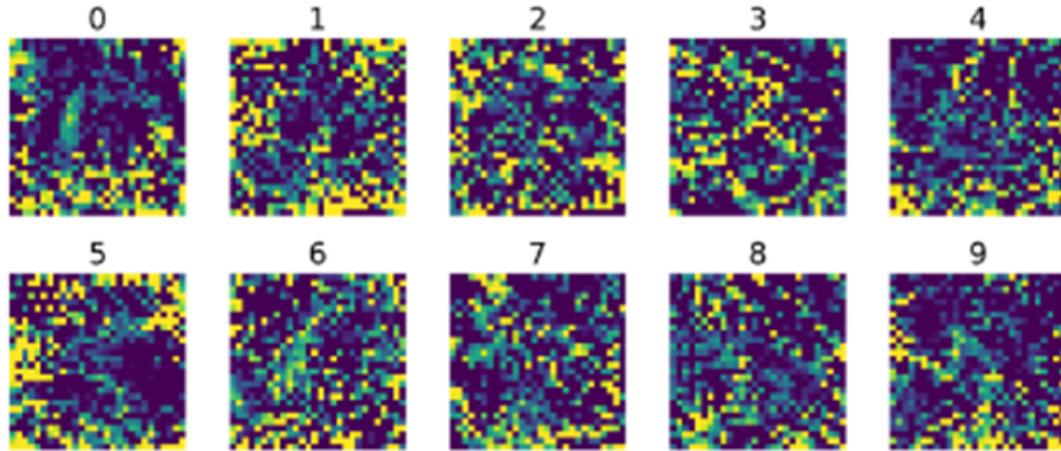


图 4: MNIST:10 幅图像训练测试精度从 12.9% 到 93.8%



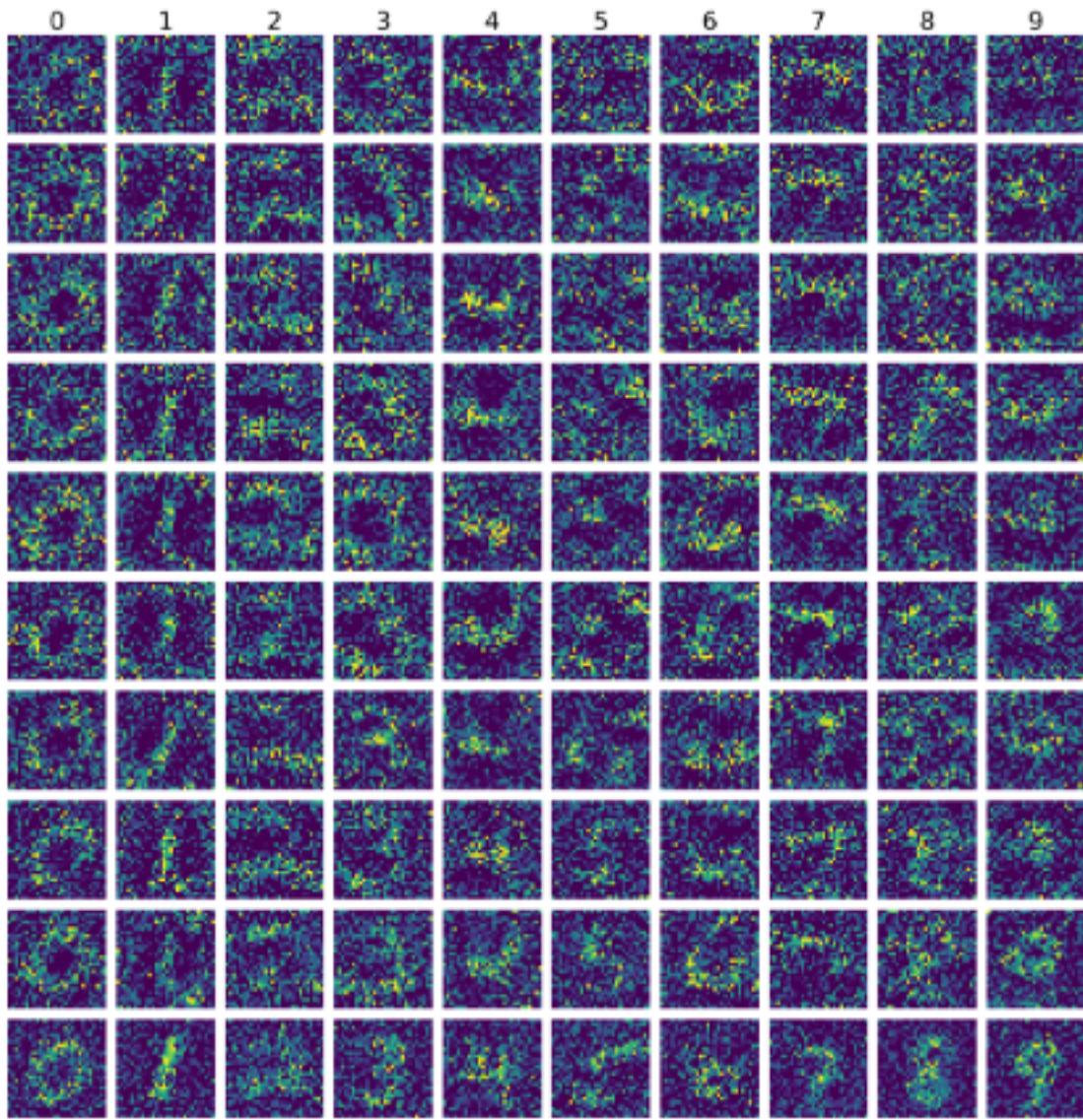


图 6: MNIST:100 张图像训练测试精度达到 $79.5\% \pm 8.1\%$

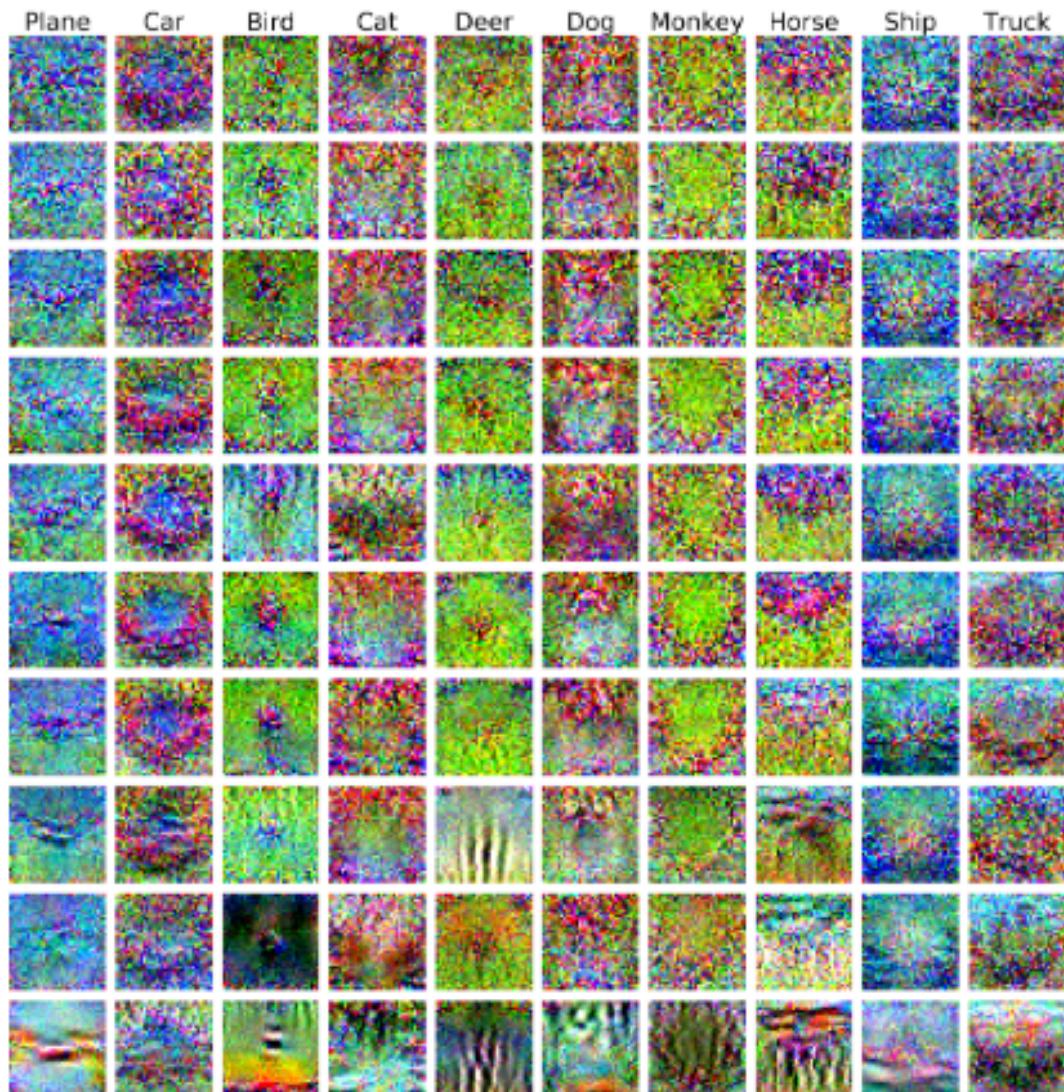


图 7: CIFAR10:100 张图像训练测试精度达到 $36.8\% \pm 1.2\%$

具有未知权重的预训练网络适应新数据集

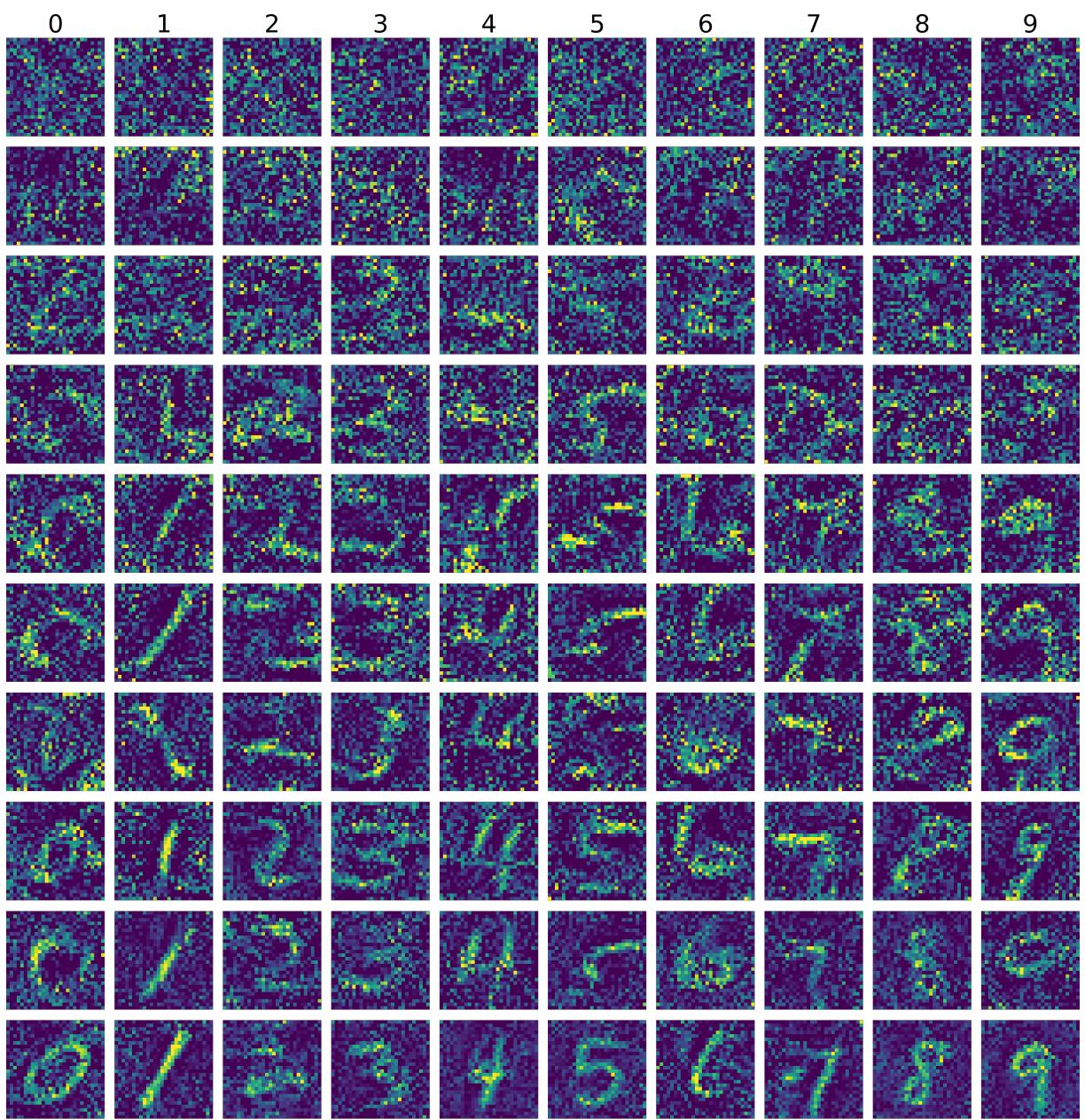


图 8: USPS->MNIST:100 张图像训练测试精度从 $67.5\% \pm 3.9\%$ 降至 $92.7\% \pm 1.4\%$



图 9: SVHN->MNIST:100 张图像训练测试精度从 $51.6\% \pm 2.8\%$ 降至 $85.2\% \pm 4.7\%$
在 1 个梯度步长内攻击具有未知权重的训练有素的分类器

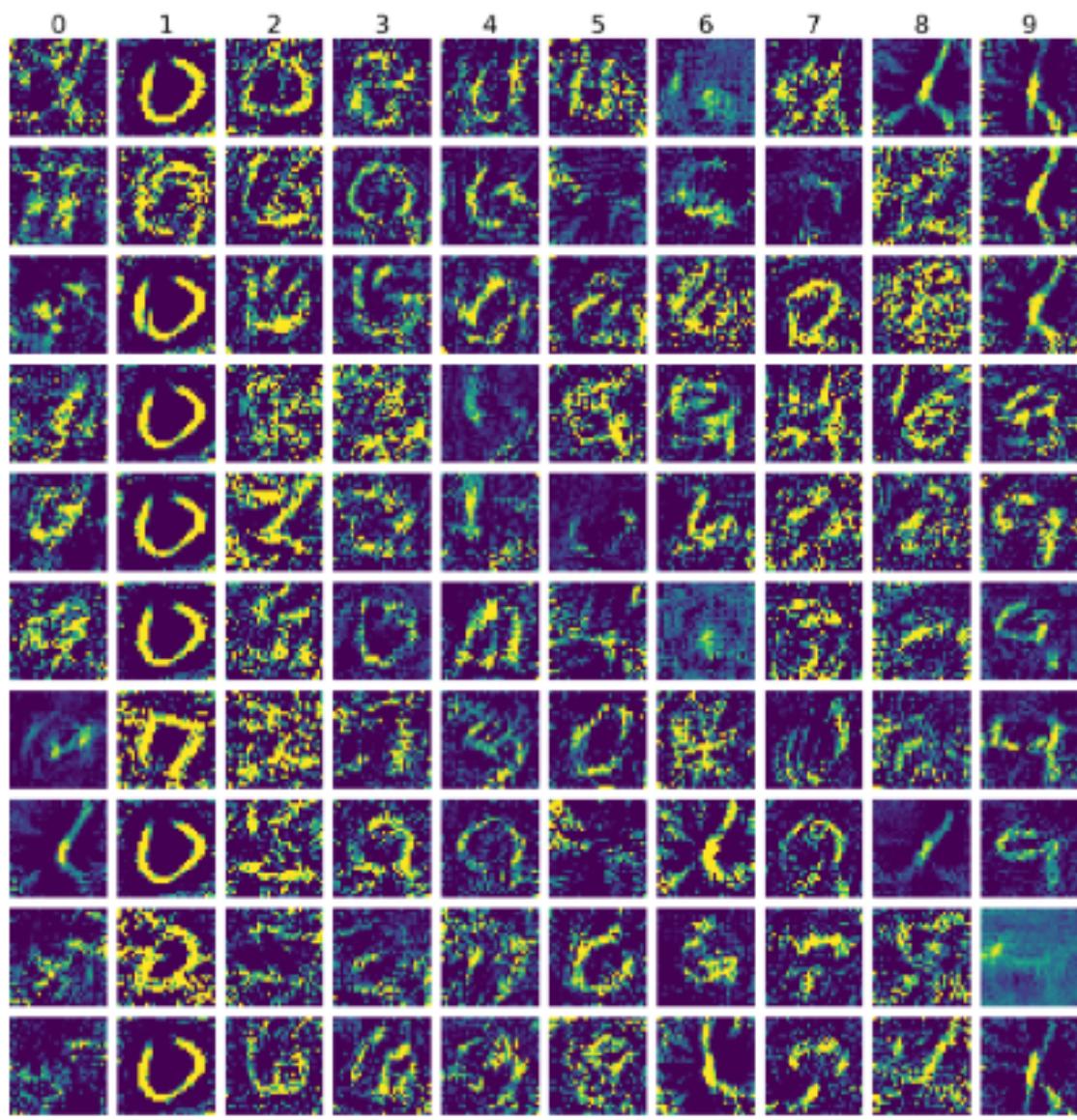


图 10: MNIST:0->1 100 张图像训练分类器，测试准确率为 $98.6\% \pm 0.5\%$ 将 $71.4\% \pm 29.6\%$ 的标签 0 测试图像预测为标签 1

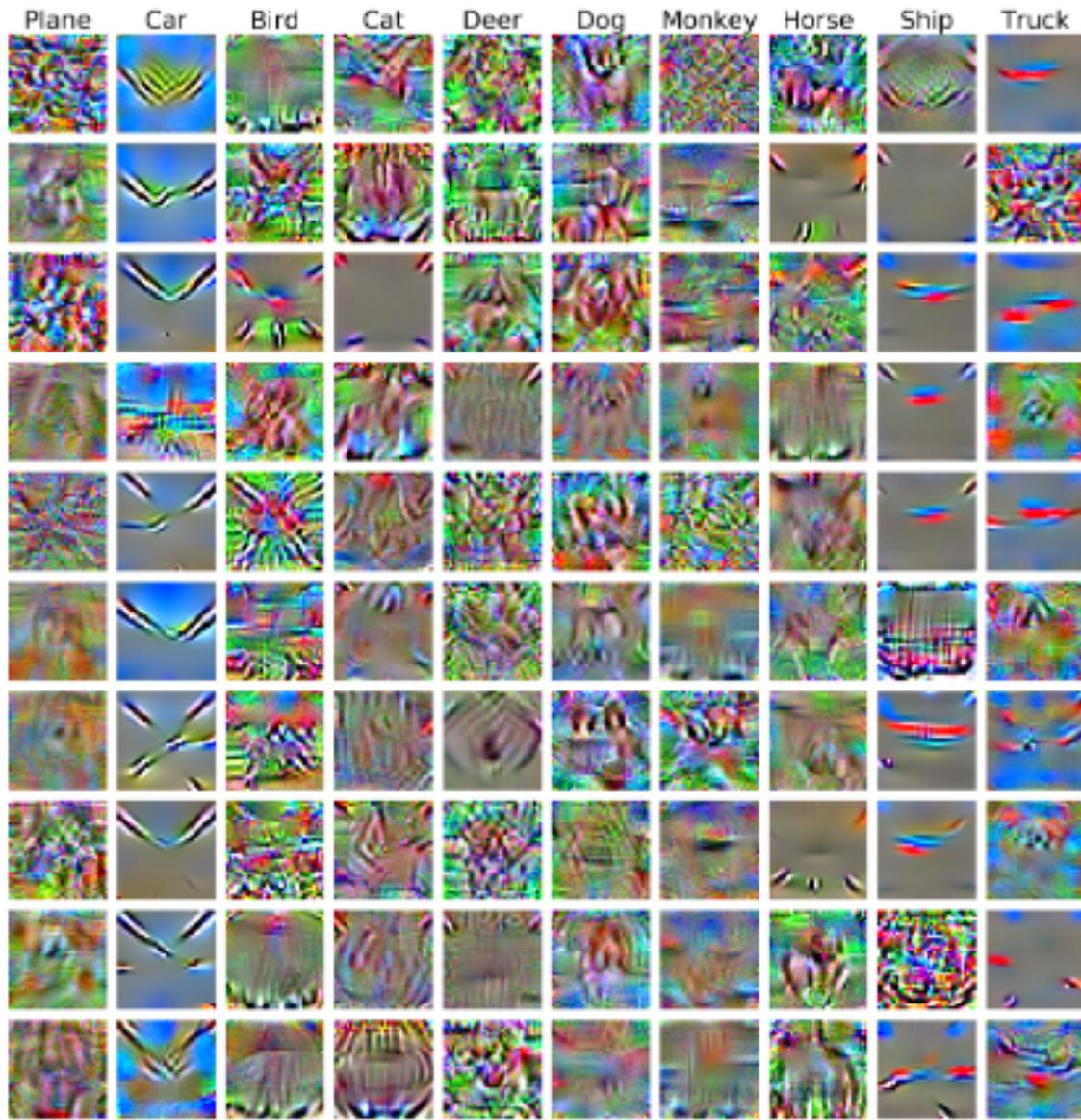


图 11: CIFAR10:plane->car 100 张图像训练分类器，测试准确率为 $78. \pm 1.1\%$ 预测 $45.9\% \pm 18.1\%$ 的标签平面测试图像作为标车

6 总结与展望

总结: 计算过程比较繁琐，在新数据集生成过程中，增加了数据集的学习率更新参数，需要更多的计算多 epoch 的情况下，需要蒸馏数据的顺序保持不变；在简单的数据集上奏效。

展望: 将数据集蒸馏应用到大规模的图片数据集（ImageNet）以及其他类型的数据上（如语音、文本），降低我们的方法对初始化的分布敏感。

参考文献

- [1] WANG T, ZHU J Y, TORRALBA A, et al. Dataset distillation[J]. arXiv preprint arXiv:1811.10959, 2018.
- [2] HINTON G, VINYALS O, DEAN J. Distilling the knowledge in a neural network[J]. arXiv preprint arXiv:1503.02531, 2015.