УДК 004.8

Нейронная сеть автокодировщик на примере CIFAR-10

Черкашин Александр Михайлович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс использования модели нейронный сети автокодировщик на примере набор данных CIFAR-10. В работе использовалась библиотека Torch, и модель автокодировщик, и в наборе данных использовался CIFAR-10 для обработки изображений. В результате работы, модель автокодировщик обрабатывает изображений, и процессе обучение получен метрика функция потери.
Ключевые слова: Автокодировщик, Сверточные нейронные сети, Torch.

Neural network autoencoder on the example of CIFAR-10

Cherkashin Alexander Mihailovich
Sholom-Aleichem Priamursky State University
student

Abstract
This article describes the process of using the autoencoder neural network model using the CIFAR-10 dataset as an example. The work used the Torch library, and the autoencoder model, and the data set used CIFAR-10 for image processing. As a result of the work, the autoencoder model processes the images, and the learning process yields a loss function metric.
Keywords: Autoencoder, Convolutional Neural Network, Torch.

1 Введение

1.1. Актуальность исследования

Актуальность исследования заключается в том что автокодировщик может удалить шум и дефекты входного изображения и повысить общую точность модели. Кроме того, модель автокодировщик можно обучить изучению основных характеристик изображения и улучшению его визуализации. Это полезно для определения определенных областей изображения, которые способствуют его классификации, и для создания новых изображений с определенными функциями.

1.2. Цель исследования

Целью работы создания и обучение модели автокодировщик для обработки изображений.

1.3. Обзор исследований

Авторы работ Щ. Щанг и др. представляет новую структуру нейронной сети, Generalized Autoencoder (генерализованный автокодировщик), для уменьшения размерности. Они объясняют, что текущие модели автокодировщик имеют ограничения в обработке нелинейных или многомерных данных, и предлагают решение, которое включает дополнительные модули для обеспечения гибкости. Авторы статьи демонстрируют эффективность своей модели посредством экспериментов с различными наборами данных, показывая повышенную точность по сравнению с традиционными автокодировщик и другими современными моделями. В целом, исследование вносит вклад в область машинного обучения и имеет потенциал для практического применения в области сжатия и визуализации данных [1].

Авторы исследователи Ц. С. Н. Патхираге и др. предлагает подход к идентификации структурных повреждений с использованием нейронных сетей автокодировщик и глубокого обучения. Исследователи провели эксперименты с использованием имитационных моделей поврежденных зданий и обнаружили, что предложенный подход позволяет точно определить местонахождение и тяжесть повреждений. В исследовании подчеркивается потенциал использования методов машинного обучения для мониторинга состояния конструкций, что может повысить безопасность и устойчивость инфраструктуры [2].

Авторы Ф. Хуанг и др. стремились разработать алгоритм глубокого обучения с использованием нейронной сети автокодировщика для прогнозирования подверженности оползням. В работе использовались данные об оползнях из китайской провинции Юньнань. Авторы использовали как экологические, так и геологические факторы в качестве входных данных для алгоритма. Они сравнили результаты своей модели с другими широко используемыми моделями для прогнозирования подверженности оползням, такими как логистическая регрессия и машины опорных векторов. В результате работы авторы показало, что нейронная сеть автокодировщика превзошла эти другие модели с точки зрения точности и точности. Авторы предполагают, что их алгоритм может быть полезен для предотвращения будущих оползней и смягчения их последствий [3].

Авторы исследования Ы. Жанг предлагает новый метод сжатия изображений с использованием сверточных автокодировщиков (CAE). Авторы показывают, что предлагаемый ими CAE превосходит традиционные автокодировщики как с точки зрения эффективности сжатия, так и с точки зрения сохранения качества изображения. Они также оценивают эффективность своего CAE при реконструкции изображений, показывая, что он дает лучшие результаты по сравнению с существующими современными методами. В целом, это исследование дает ценную информацию об улучшении методов сжатия и реконструкции изображений с использованием CAE [4].

Авторы статьи Г. Тодерици и др. предложили новый подход к сжатию изображений с использованием рекуррентных нейронных сетей (RNN). В

частности, они представляют структуру сжатия на основе RNN с переменной скоростью, которая способна достигать высоких коэффициентов сжатия при сохранении качества изображения. Они демонстрируют эффективность своего подхода посредством экспериментов с несколькими наборами данных изображений, показывая, что он превосходит существующие методы сжатия с точки зрения, как эффективности сжатия, так и качества восприятия. В целом, это исследование представляет собой многообещающее направление для разработки более эффективных методов сжатия изображений на основе RNN [5].

Авторы исследователи П. Р. Дачапаллы рассматривает использования сверточных нейронных сетей (CNN) и репрезентативных единиц автоэнкодера (RAU) для обнаружения эмоций по выражениям лица. Исследователи использовали набор данных, состоящий из 8 различных эмоций, и обучили модель CNN с RAU для точного обнаружения этих эмоций. Исследование показало, что предложенный метод превзошел предыдущие методы обнаружения эмоций, достигнув точности до 91,69%. В этом исследовании представлен многообещающий подход к обнаружению эмоций по выражению лица, который может иметь различные применения в таких областях, как психология, реклама и взаимодействие человека с компьютером [6].

2. Рабочий процесс

2.1. Набор данных

Исходные данные использовался CIFAR-10 [7]. В исходные данные представлено серия изображения размера 32x32.

Обработка изображения выполнялось только преобразования в тензор.

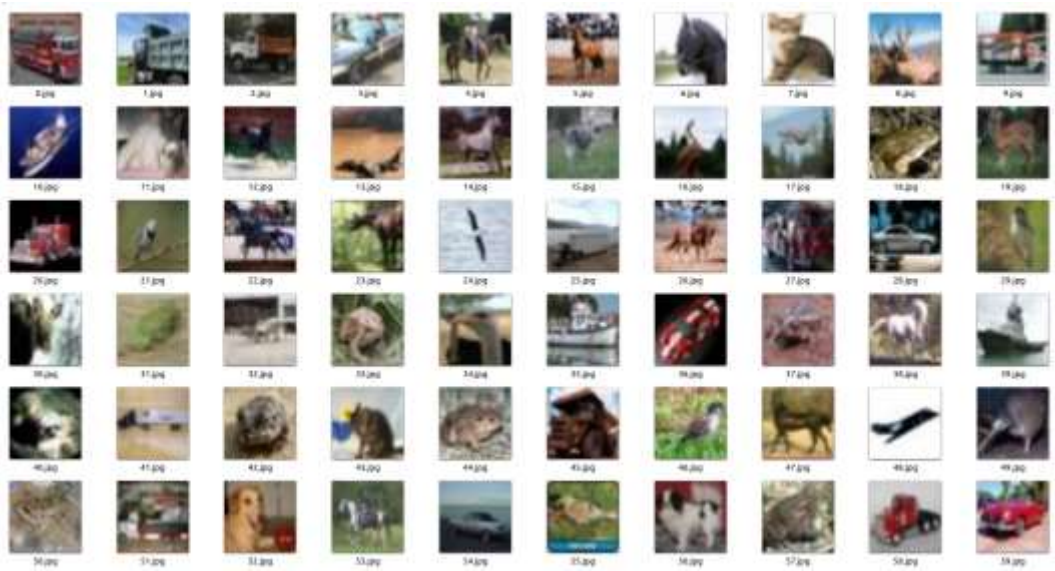

Рисунок 1. Набор данных обучающийся признаков

2.1. Модель

Модель нейронной сети автокодировщик — это модель обучения без учителя при использовании метода обратного распространения ошибки,

которая берет набор входных данных и обучается, как лучше всего представить их в сжатой форме. Затем сжатая форма используется для восстановления исходных входных данных. Его архитектура состоит из двух основных частей: кодировщик и декодировщик. Кодировщик берет входные данные и сжимает их в представление меньшего размера, в то время как декодировщик пытается восстановить исходные данные из сжатой формы.

В целом, модель нейронной сети автокодировщик представляет собой мощный инструмент для извлечения признаков и обучения без учителя с многочисленными потенциальными [8].

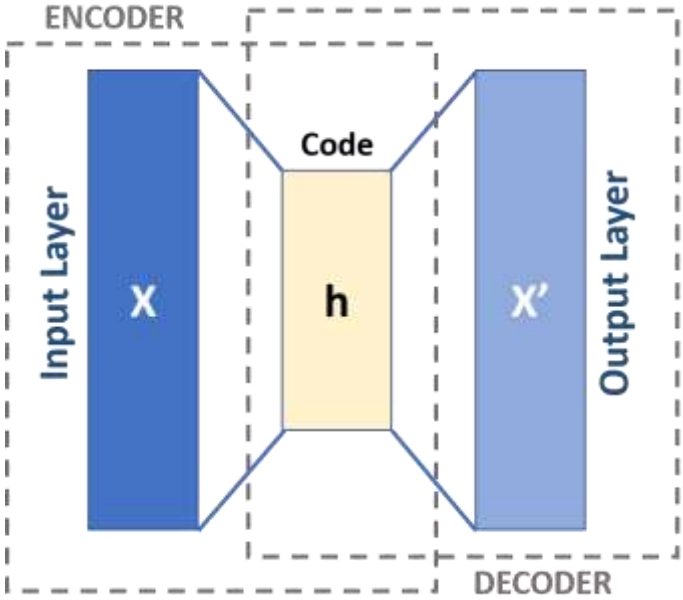

Рисунок 2. Схема автокодировщик

Input Layer — входной слой
Output Layer — выходной слой
Encoder — кодировщик
Code — код
Decoder — декодировщик (рис 2).

Листинг 2.1. Структура модели автокодировщик.

```
ConvAutoencoder(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (t_conv1): ConvTranspose2d(4, 16, kernel_size=(2, 2), stride=(2, 2))
  (t_conv2): ConvTranspose2d(16, 3, kernel_size=(2, 2), stride=(2, 2))
  (relu): ReLU()
  (sigmoid): Sigmoid()
)
```

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1         [128, 16, 32, 32]             448
            ReLU-2         [128, 16, 32, 32]               0
```

```
      MaxPool2d-3        [128, 16, 16, 16]             0
          Conv2d-4        [128, 4, 16, 16]           580
            ReLU-5        [128, 4, 16, 16]             0
       MaxPool2d-6          [128, 4, 8, 8]             0
  ConvTranspose2d-7       [128, 16, 16, 16]           272
            ReLU-8       [128, 16, 16, 16]             0
  ConvTranspose2d-9        [128, 3, 32, 32]           195
       Sigmoid-10        [128, 3, 32, 32]             0
================================================================
Total params: 1,495
Trainable params: 1,495
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.50
Forward/backward pass size (MB): 52.25
Params size (MB): 0.01
Estimated Total Size (MB): 53.76
----------------------------------------------------------------
```

Листинг 2.2. Параметры для обучения модели

```
1  self.criterion = nn.BCELoss().to(self.device)
2  self.lr = 0.001
3  self.model = {
4          "encoder": ConvAutoencoder().to(self.device)
5  }
6  self.optimizer = {
7          "encoder": torch.optim.Adam(self.model["encoder"].parameters(), lr=self.lr)
8  }
```

Строка 1. Метрика BCELoss.
Строка 2. Скорость обучения 0.001.
Строка 3 — 5. Модель Autoencoder.
Строка 6 — 8. Оптимизатор Adam.

2.2. Обучение

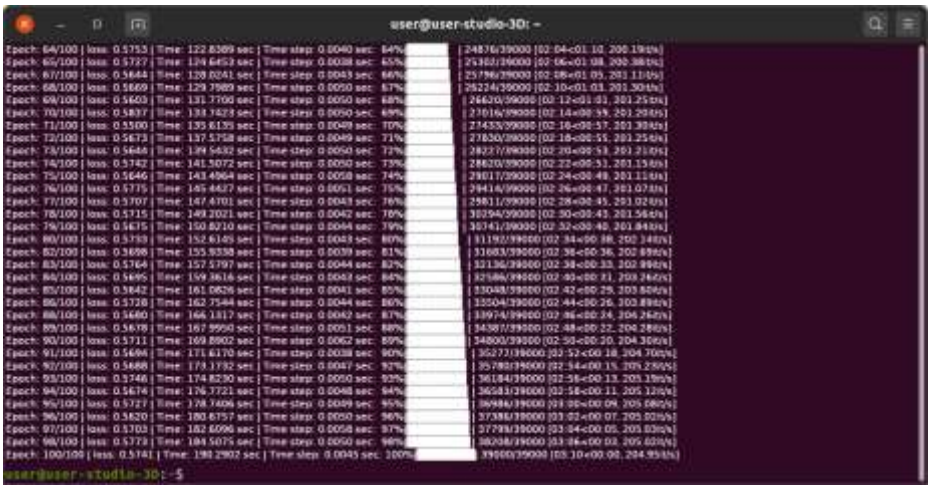

Рисунок 3. Обучение модели

В результате обучение последний цикл обучение метрику функция потери удалось достигнуть 0,574. Однако в программе задано эпоха обучения

100, размер пакетов 128, оценка модели использовалась метрика BCELoss. Программа написано на языке Python использует библиотека Torch.

Листинг 2.3. Функция для обучение модель

```
1  def step_train(self, sel: any, index :int) -> None:
2          super().step_train(sel, index)
3          images, _ = sel
4          images = images.to(self.device)
5          self.optimizer["encoder"].zero_grad()
6          outputs = self.model["encoder"](images)
7          loss = self.criterion(outputs, images)
8          loss.backward()
9          self.optimizer["encoder"].step()
10         self.metric.loss = loss.item()
```

Строка 2, заглушка.

Строка 3 — 4, загрузка данных, изображений.

Строка 5, обнуление градиентам, очищает градиенты всех оптимизированных переменных.

Строка 6, предсказания модель, выполняет прогнозируемых выходных данных

Строка 7, оценка модель метрика BCELoss

Строка 8, вычисляет обратный проход градиент функция потерь по отношению к параметрам модели.

Строка 9 выполняет шаг обновление параметра модели, корректирует весов нейронной сети в направлении минимизации функции потерь.

Строка 10. сохраняем метрику в список.

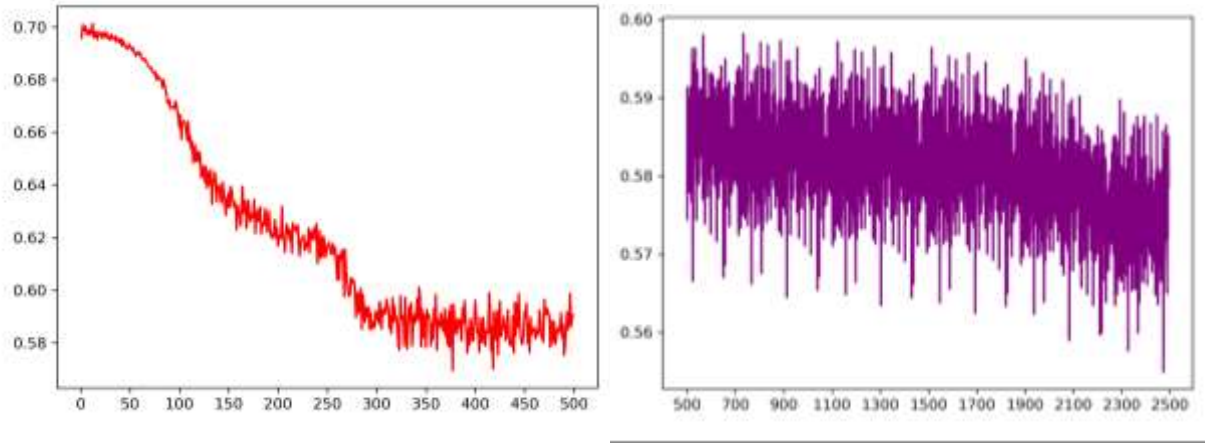

Рисунок 4. Функция потерь (метрика BCELoss)

На графике представлено по оси X цикл обучения, а Y — метрика BCELoss функция потерь быстро убывала до цикла 300 (значение потерь 0.592), а затем медленно убывает до цикла 2500 (значение потерь 0,576), функция потерь, после 2500 циклов — очень медленно, на графике представлено уменьшение скорость убывания, начиная 2500 циклов до 39000 циклов (рис 4).

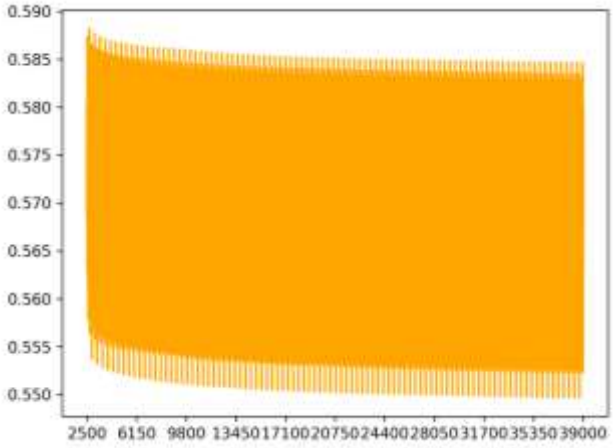

Рисунок 5. Функция потерь (метрика BCELoss)

Начальная шаг обучения значение функция потерь была 0,696 а до конца цикла обучения потерь была 0,574, минимальная точка функция потерь — 0,54966, цикл обучения была 38743.

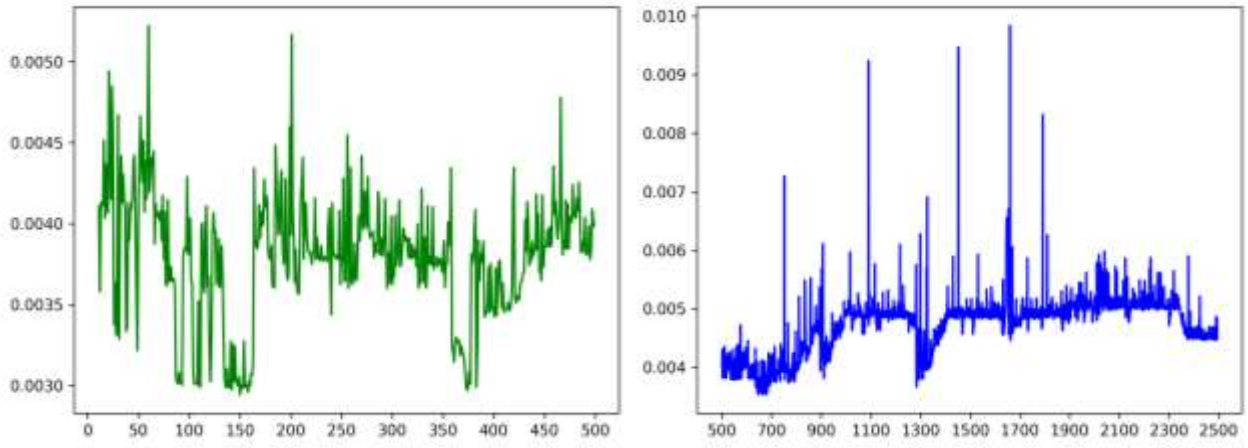

Рисунок 6. Время обучения

Время (в секундах) обучения представлен на графике от 0 до 500 циклов и 500 до 2500 циклов с учетом задержки загрузки и обработки набор данных, однако обработки данных выполнялся в начале цикла обучение а затем последующий загрузки использовался кэш чтобы сократить вычислительные сложности.

Время выполнение обучение ушло 178,987 секунда (2 минута, 58,987 секунда).

2.3. Предсказание

Рисунок 7. Слева исходное изображения, справа обработано моделью автокодировщиком

Рисунок 8. Слева исходное изображения, справа обработано моделью автокодировщиком

Рисунок 9. Слева исходное изображения, справа обработано моделью автокодировщиком

Модель автокодировщик исправляет цвета изображений (рис 7, рис 8, рис 9).

Листинг 2.4. Использования модель автокодировщик

```
1  app.model["encoder"].to("cpu").eval()
2  sel_image_org = app.load_image("./lena.png")
```

```
3 sel_img = app.transform(sel_image_org)
4 img = app.model["encoder"](sel_img)
5 save_image(img, "./out.jpg")
```

Строка 1 использования модель в режим исполнения.
Строка 2 загрузка изображения.
Строка 3 преобразования изображения в тензор.
Строка 4 выполнение модель предсказания.
Строка 5 вывод изображения в файл.

3 Выводы

В данное статье была использована модель автокодировщик для улучшения качества изображения путем исправления цвета и шумоподавления, однако в данной статьи использовался набор данных CIFAR-10 для обучения признаков, в результаты работы обучения модель автокодировщика удалось достичь функция потерь минимального значения 0,54966 при цикле обучения 38743.

Библиографический список

1. Wang W. et al. Generalized autoencoder: A neural network framework for dimensionality reduction //Proceedings of the IEEE conference on computer vision and pattern recognition workshops. 2014. С. 490-497.
2. Pathirage C. S. N. et al. Structural damage identification based on autoencoder neural networks and deep learning //Engineering structures. 2018. Т. 172. С. 13-28.
3. Huang F. et al. A deep learning algorithm using a fully connected sparse autoencoder neural network for landslide susceptibility prediction //Landslides. 2020. Т. 17. С. 217-229.
4. Zhang Y. A better autoencoder for image: Convolutional autoencoder //ICONIP17-DCEC. Available online: http://users. cecs. anu. edu. au/Tom. Gedeon/conf/ABCs2018/paper/ABCs2018_paper_58. pdf (accessed on 23 March 2017). 2018.
5. Toderici G. et al. Variable rate image compression with recurrent neural networks //arXiv preprint arXiv:1511.06085. 2015.
6. Dachapally P. R. Facial emotion detection using convolutional neural networks and representational autoencoder units //arXiv preprint arXiv:1706.01509. 2017.
7. CIFAR-10 and CIFAR-100 datasets // CIFAR-100 datasets URL: https://www.cs.toronto.edu/~kriz/cifar.html (дата обращения: 2023-05-02).
8. Autoencoder - Wikipedia // Wikipedia URL: https://en.wikipedia.org/wiki/Autoencoder (дата обращения: 2023-05-03).

4. Приложения

Листинг 4.1. Исходный код программы

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
from torchsummary import summary
from torchvision import transforms, datasets
import torchvision.transforms.functional as F
from typing import Optional, Callable, TypeVar, Type, Union
from PIL import Image
from Lib.AppMain import *
from torch.cuda import amp
import torch.nn.functional as nnf
import matplotlib.pyplot as plt
from collections import deque
import os

class ConvAutoencoder(nn.Module):
	def __init__(self):
		super(ConvAutoencoder, self).__init__()
		self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
		self.conv2 = nn.Conv2d(16, 4, kernel_size=3, padding=1)
		self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

		self.t_conv1 = nn.ConvTranspose2d(4, 16, kernel_size=2, stride=2)
		self.t_conv2 = nn.ConvTranspose2d(16, 3, kernel_size=2, stride=2)
		self.relu = nn.ReLU()
		self.sigmoid = nn.Sigmoid()
	def forward(self, x):
		x = self.relu(self.conv1(x))
		x = self.pool(x)
		x = self.relu(self.conv2(x))
		x = self.pool(x)
		x = self.relu(self.t_conv1(x))
		x = self.sigmoid(self.t_conv2(x))
		return x

class App(AppMain):
	def file_begin(self, fname :str, mode :str) -> any:
		return open(self.file_name(fname), mode)
	def file_name(self, fname :str) -> str:
		return fname
	def main(self):
		self.dir_prefix = "./Data"
		self.dirs = {
			"dataset": "cifar-10",
			"fig": "Fig",
			"viewt": "View",
			"view_test": "View_test"
		}
		self.profile_name = "default"
		self.datasets = deque()
		self.model = {}
		self.optimizer = {}
		self.auto_save_exit = False
		self.init_dirs()
		self.disp_metric = ["loss"]
		self.metric.loss  = None
		self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
		self.criterion = nn.BCELoss().to(self.device)
		self.lr = 0.001
		self.model = {
			"encoder": ConvAutoencoder().to(self.device)
		}
		self.optimizer = {
			"encoder": torch.optim.Adam(self.model["encoder"].parameters(), lr=self.lr)
		}
		self.transform = transforms.ToTensor()
		self.cache_data = True
		self.init_data()
	def init_data(self):
		train_data = datasets.CIFAR10(root=app.get_path("dataset"), train=True,
```

```
                                                    download=False, transform=self.transform)
                test_data = datasets.CIFAR10(root=app.get_path("dataset"), train=False,
                                                    download=False, transform=self.transform)

                self.datasets.append(Datasets_batch(train_data, batch_size=128))
                self.datasets.append(Datasets_batch(test_data, batch_size=128))
                self.init_datasets(num_workers=0)
                self.cache_data = True
        def start_train(self) -> None:
                self.model["encoder"].train()
        def step_train(self, sel: any, index :int) -> None:
                super().step_train(sel, index)
                images, _ = sel
                images = images.to(self.device)
                self.optimizer["encoder"].zero_grad()
                outputs = self.model["encoder"](images)
                loss = self.criterion(outputs, images)
                loss.backward()
                self.optimizer["encoder"].step()
                self.metric.loss = loss.item()
        def load_image(self, path :str) -> Image.Image:
                with open(path, "rb") as f:
                        img = Image.open(f)
                        return img.convert("RGB")
        def save_image(self, path :str, img :Union[np.ndarray, Image.Image]) -> None:
                pic = None
                with open(path, "wb") as f:
                        if type(img) != Image.Image:
                                pic = Image.fromarray(img, "RGB")
                        else:
                                pic = img
                        pic.save(f)

app = App()
app.main()

i = 0
for images, labels in app.datasets[0].dataLoader:
        p = app.get_path("view")
        for j in range(len(images)):
                cl = p + "/"
                img_d = F.to_pil_image(images[j])
                app.save_image(cl + "/" + "%s.jpg" % i, img_d)
                i += 1

i = 0
for images, labels in app.datasets[1].dataLoader:
        p = app.get_path("view_test")
        for j in range(len(images)):
                cl = p + "/"
                img_d = F.to_pil_image(images[j])
                app.save_image(cl + "/" + "%s.jpg" % i, img_d)
                i += 1


summary(app.model["encoder"], input_size=(3, 32, 32), batch_size=128)

app.fit(100)
app.save_model("encoder")

#----------------------------

app.load_model("encoder")

app.model["encoder"].to("cpu").eval()

url_image = "./images"

sel_image_org = app.load_image(url_image + "/lena.png")
sel_img = app.transform(sel_image_org)
img = app.model["encoder"](sel_img)
save_image(img, url_image + "/out.jpg")

# Plot

df = pd.DataFrame(app.metric._data)
df_m = df[:500]
df_mt = df[10:500]
```

```
df_ml = df[500:2500]
df_ml2 = df[2500:]

df_m2 = df[500:]
x_step = 50

#Time

fig, ax = plt.subplots()
ax.plot(df_mt["Step"], df_mt["Time"], label ='Time', color="green")
ax.set_xticks(np.arange(0, max(df_mt["Step"])+2, x_step))
fig.savefig(app.get_path("fig", "{0}.png".format("Time")), dpi = 300)

x_step = 200

fig, ax = plt.subplots()
ax.plot(df_ml["Step"], df_ml["Time"], label ='Time', color="blue")
ax.set_xticks(np.arange(500, max(df_ml["Step"])+10, x_step))
fig.savefig(app.get_path("fig", "{0}.png".format("Time_long")), dpi = 300)

x_step = 3650

fig, ax = plt.subplots()
ax.plot(df_ml2["Step"], df_ml2["Time"], label ='Time', color="pink")
ax.set_xticks(np.arange(2500, max(df_ml2["Step"])+10, x_step))
fig.savefig(app.get_path("fig", "{0}.png".format("Time_long2")), dpi = 300)

# Loss

fig, ax = plt.subplots()
ax.plot(df_m["Step"], df_m["loss"], label ='loss', color="red")
ax.set_xticks(np.arange(0, max(df_m["Step"])+2, x_step))
fig.savefig(app.get_path("fig", "{0}.png".format("loss")), dpi = 300)

#x_step = 3850
x_step = 200

fig, ax = plt.subplots()
ax.plot(df_ml["Step"], df_ml["loss"], label ='loss', color="purple")
ax.set_xticks(np.arange(500, max(df_ml["Step"])+10, x_step))
fig.savefig(app.get_path("fig", "{0}.png".format("loss_long")), dpi = 300)

x_step = 3650

fig, ax = plt.subplots()
ax.plot(df_ml2["Step"], df_ml2["loss"], label ='loss', color="orange")
ax.set_xticks(np.arange(2500, max(df_ml2["Step"])+10, x_step))
fig.savefig(app.get_path("fig", "{0}.png".format("loss_long2")), dpi = 300)

df["loss"].idxmin()
```