УДК 004.8

Классификация одежды с набором данных MNIST

Черкашин Александр Михайлович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описан процесс использования модели нейронный сети для определения класса изображений. В работе использовалась библиотека Torch, и модель CNN, и представленный набор данных MNIST, классу изображений с разметкой текста. В результате работы была оценена модель способность определять класс изображения, а также оценена точность и функция потерь.
Ключевые слова: MNIST, сверточные нейронные сети, классификация изображения, Torch.

Clothing classification with MNIST dataset

Cherkashin Alexander Mihailovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes the process of using a neural network model to define a class of images. The work used the Torch library, and the CNN model, and the presented MNIST dataset, a class of images with text markup. As a result of the work, the model's ability to determine the image class was evaluated, as well as the accuracy and loss function were evaluated.
Keywords: MNIST, Convolutional Neural Network, Image classification, Torch.

1 Введение

1.1. Актуальность исследования

Актуальность исследования заключается в классификация изображений одежды становится все более актуальной в сфере компьютерного зрения и машинного обучения. Алгоритмы классификации также можно использовать для анализа модных тенденций, рекомендаций по размеру и индивидуального стиля. Кроме того, классификация изображений одежды она может помочь идентифицировать людей на основе их выбора одежды. Таким образом, точная классификация изображений одежды обладает значительным потенциалом, чтобы произвести революцию в различных отраслях и улучшить повседневную жизнь.

1.2. Цель исследования

Целью работы является создание и обучение модели для классификации одежды на основе набора данных MNIST.

1.3. Обзор исследований

Н. Шохам и др. предлагает новый метод, включающий в себя обучающие модели с использованием как централизованных данных, так и локальных данных с отдельных устройств. Исследование демонстрирует, как этот подход может помочь повысить точность, а также решить проблемы с конфиденциальностью. Они протестировали свою методологию на нескольких наборах данных, таких как MNIST, CIFAR-10 и MNIST-incremental, и обнаружили, что их подход превосходит другие традиционные методы. В целом, исследование представляет собой многообещающий подход к преодолению забывания в машинном обучении [1].

Щ. Женг и др. оценивает свой подход с использованием реального приложения для глубокого обучения и показывают, что он превосходит существующие методы распределения ресурсов с точки зрения, как производительности приложения, так и эффективности использования ресурсов. В целом, исследование представляет собой эффективную стратегию эффективного распределения ресурсов в системах с несколькими арендаторами для приложений глубокого обучения [2].

Авторы Ю. Ли и Х. Лю. сосредоточены на реализации стохастического квазиньютоновского метода в Pytorch, популярной библиотеке машинного обучения с открытым исходным кодом. Авторы подробно объясняют алгоритм и то, как его можно использовать для повышения эффективности обучения глубоких нейронных сетей. Они также представляют экспериментальные результаты, демонстрирующие эффективность их реализации на различных эталонных наборах данных. В целом, исследование предоставляет ценный ресурс для исследователей и практиков, заинтересованных в оптимизации производительности моделей машинного обучения [3].

Авторы Х. Хиао, К. Расул, Р. Воллграф представляют новый набор данных изображений под названием Fashion-MNIST, который можно использовать для сравнительного анализа алгоритмов машинного обучения. Набор данных состоит из 70 000 изображений в оттенках серого различных предметов одежды, таких как обувь, сумки и рубашки. Авторы работ предоставляют некоторые первоначальные результаты, сравнивая производительность различных алгоритмов классификации в наборе данных Fashion-MNIST, и показывают, что он может обеспечить сложный эталон для оценки моделей машинного обучения. В целом, исследование является полезным вкладом в области машинного обучения и компьютерного зрения [4].

А. Ёусефпоур, представляют Opacus, удобную библиотеку дифференциальной конфиденциальности в PyTorch. Библиотека предоставляет несколько механизмов для обучения моделей машинного

обучения. Он также предлагает простой и интуитивно понятный API, позволяющий пользователям легко включать в свои конвейеры машинного обучения. Исследование демонстрирует эффективность Opacus посредством экспериментов с различными наборами данных, и результаты показывают, что он может обеспечить надежные сохранении точности модели [5].

М. МсКенна сравнивал различные типы функций активации для глубокого обучения в наборе данных Fashion-MNIST. Результаты показали, что функция активации Swish превзошла обычно используемые функции ReLU и Leaky ReLU, когда речь идет о точности и скорости сходимости. Исследование предполагает, что использование функции активации Swish может повысить производительность моделей глубокого обучения в задачах распознавания изображений [6].

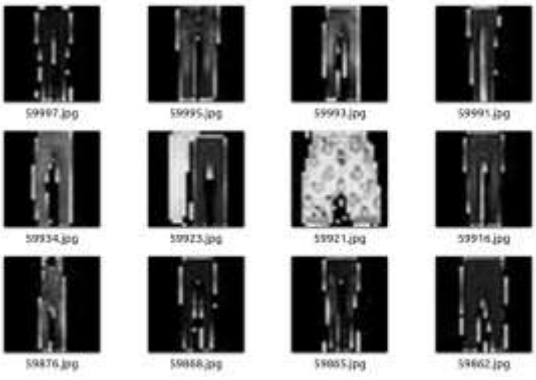
2. Рабочий процесс

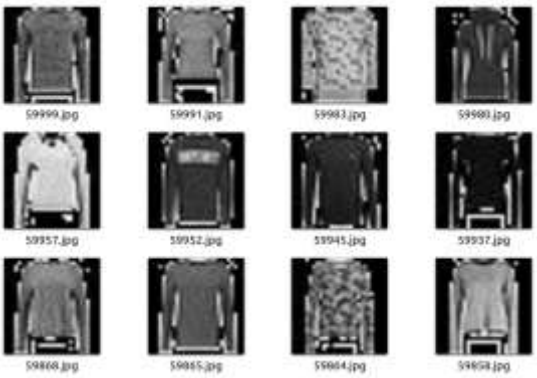
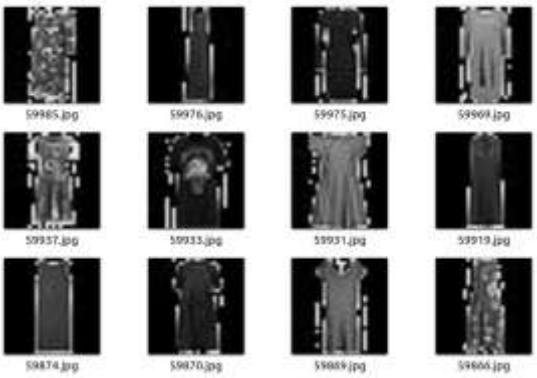
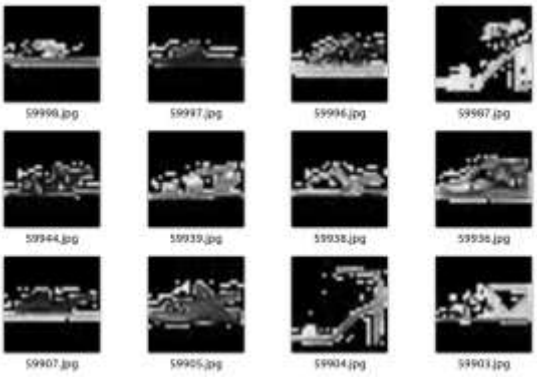
2.1. Набор данных

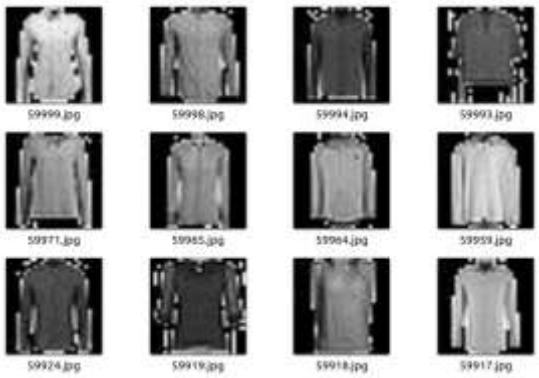
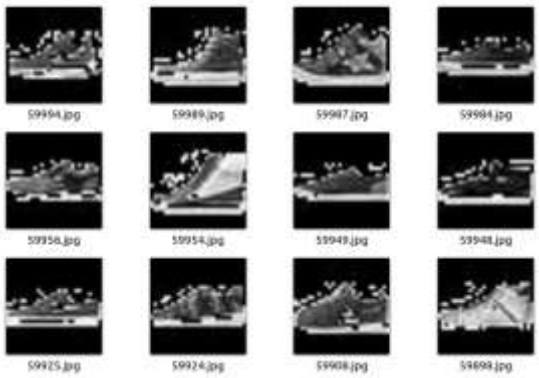
Исходные данные представленный MNIST. В исходные данные представлено серия изображения с разметкой, каждый изображения поделен на классы, изображения содержится одежды [7].

Размер пакетов для обучения серий изображений (batch_size) 100. Разрешение изображения 28x28 пикселей.

Таблица 1. Набор данных, изображения по классу

Названия класса	Метка (класс)	Набор данных
T-shirt/Top	0	
Trouser	1	

Pullover	2	
Dress	3	
Coat	4	
Sandal	5	

Shirt	6	
Sneaker	7	
Bag	8	
Ankle Boot	9	

Листинг 2.1. Набор действия для обработки набор данных.

```
1 self.transform = transforms.Compose([
2         transforms.ToTensor()
3 ])
```

Строка 2. Преобразования изображений в тензор.

2.1. Модель

Вход в модель подается изображения, а на выход числовое значение класса изображений.

Листинг 2.2. Структура модели CNN.

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1         [100, 32, 28, 28]             320
     BatchNorm2d-2         [100, 32, 28, 28]              64
            ReLU-3         [100, 32, 28, 28]               0
       MaxPool2d-4         [100, 32, 14, 14]               0
          Conv2d-5         [100, 64, 12, 12]          18,496
     BatchNorm2d-6         [100, 64, 12, 12]             128
            ReLU-7         [100, 64, 12, 12]               0
       MaxPool2d-8           [100, 64, 6, 6]               0
          Linear-9                [100, 600]       1,383,000
       Dropout2d-10                [100, 600]               0
         Linear-11                [100, 120]          72,120
         Linear-12                 [100, 10]           1,210
================================================================
Total params: 1,475,338
Trainable params: 1,475,338
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.30
Forward/backward pass size (MB): 86.07
Params size (MB): 5.63
Estimated Total Size (MB): 92.00
----------------------------------------------------------------
```

```
FashionCNN(
  (layer1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Linear(in_features=2304, out_features=600, bias=True)
  (drop): Dropout2d(p=0.25, inplace=False)
  (fc2): Linear(in_features=600, out_features=120, bias=True)
  (fc3): Linear(in_features=120, out_features=10, bias=True)
)
```

layer1 и **layer2** — сверточная нейронная сеть для изображения.

fc1 — линейная нейронная сеть для классификация изображения, на выходе выводит 10 классов.

Листинг 2.3. Параметры для обучения модели

```
1  self.criterion = nn.CrossEntropyLoss().to(self.device)
2  self.lr = 0.001
3  self.model = {
4          "FashionCNN": FashionCNN().to(self.device)
5  }
6  self.optimizer = {
```

```
7           "FashionCNN": torch.optim.Adam(self.model["FashionCNN"].parameters(), lr=self.lr)
8   }
```

Строка 1. Метрика cross entropy.
Строка 2. Скорость обучения.
Строка 3 — 5. Модель FashionCNN.
Строка 6 — 8. Оптимизатор Adam.

2.2. Обучение

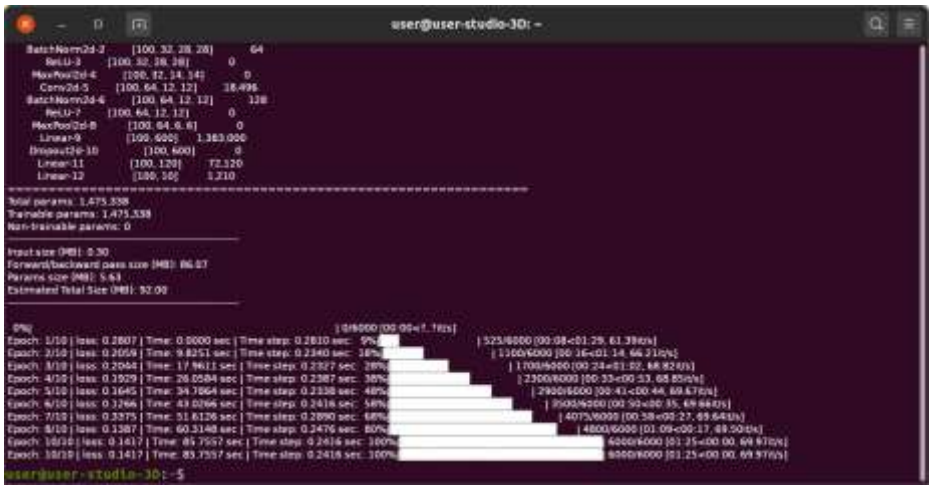


Рисунок 1. Обучение модели

Программа написано на языке Python использовалась библиотека Torch. Программа выполняет обучение модели (рис 1), задано эпохи 10, скорость обучение 0.001.

Для оценки модели использовали метрику CrossEntropyLoss для обучения.

Листинг 2.4. Функция для обучения модели

```
1   def step_train(self, sel: any, index :int) -> None:
2           super().step_train(sel, index)
3           model = self.model["FashionCNN"]
4           optimizer = self.optimizer["FashionCNN"]
5           images, labels = sel
6           train = Variable(images.view(100, 1, 28, 28)).to(self.device)
7           labels = Variable(labels).to(self.device)
8           outputs = model(train)
9           loss = self.criterion(outputs, labels)
10          optimizer.zero_grad()
11          loss.backward()
12          optimizer.step()
13          if not (self.metric.Step % 25):
14                  total = 0
15                  correct = 0
16                  for images, labels in self.datasets[1].dataLoader:
17                          images, labels = images.to(self.device), labels.to(self.device)
18                          test = Variable(images.view(100, 1, 28, 28))
19                          outputs = model(test)
20                          predictions = torch.max(outputs, 1)[1].to(self.device)
21                          correct += (predictions == labels).sum()
22                          total += len(labels)
23                  accuracy = correct * 100 / total
```

```
24                  self.metric.accuracy = accuracy
25                  self.metric.predictions = predictions
26                  self.metric.labels = labels
27          else:
28                  self.metric.accuracy = None
29                  self.metric.predictions = None
30                  self.metric.labels = None
31          self.metric.loss = loss.item()
```

Строка 2, заглушка.
Строка 3 — 7, получение перемены для обучения.
Строка 8 — 12. обучаем модель.
Строка 13 — 30. оценка модель по тестовому данных метрика acc.
Строка 31. сохраняем метрику в список.

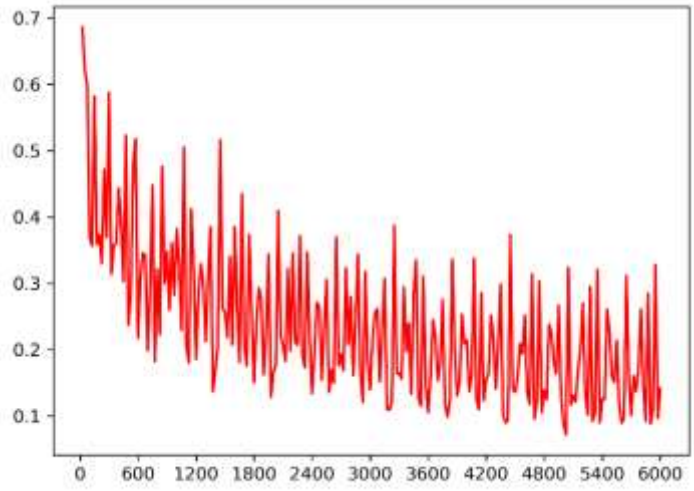


Рисунок 2. Функция потерь (метрика cross entropy)

В результате обучения, мы получили значение функций потерь (по оси Y) при 6000 циклов, минимальная функция потерь была достигнута 0,0717 (по оси X) (рис 2).

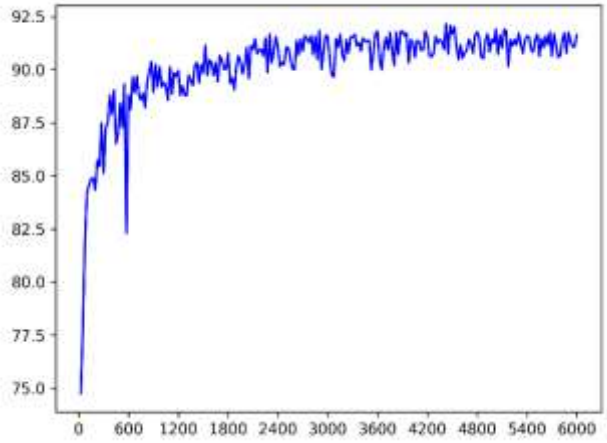


Рисунок 3. Метрика Acc

Метрика Acc (accuracy) оценка сходство данные полученный модели с тестовый данный. Высокие значение — модель лучшее обобщает тестовый данные, максимальное значение точность достигла 92,15.

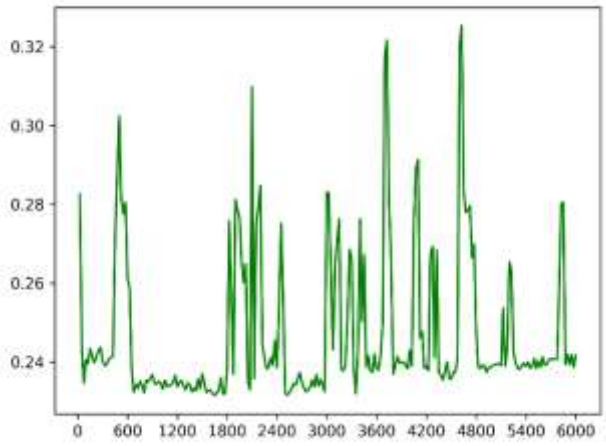


Рисунок 4. Время обучения

Время обучение на графике представлено задержка шаг выполнения обучения с учетом каждый 25 шаг выполнялось оценка модели по тестовому данных.

Время выполнение обучение было 87,139 секунда (1 минута, 27,139 секунда).

2.3. Предсказание

Таблица 2. Оценка модели

Название класса	Метрика точность (Accuracy).
T-shirt/Top	88.40%
Trouser	99.40%
Pullover	88.80%
Dress	85.90%
Coat	90.90%
Sandal	98.40%
Shirt	74.50%
Sneaker	93.10%
Bag	97.80%
Ankle Boot	97.80%

Модель лучшее предсказывает точность по классам Trouser и Sandal.

Листинг 2.5. Оценка модели метрика Accuracy.

```
class_correct = [0. for _ in range(10)]
total_correct = [0. for _ in range(10)]

with torch.no_grad():
        for images, labels in app.datasets[1].dataLoader:
                images, labels = images.to(app.device), labels.to(app.device)
                test = Variable(images)
                outputs = app.model["FashionCNN"](test)
                predicted = torch.max(outputs, 1)[1]
                c = (predicted == labels).squeeze()

                for i in range(100):
                        label = labels[i]
                        class_correct[label] += c[i].item()
                        total_correct[label] += 1

for i in range(10):
        print("Accuracy of {}: {:.2f}%".format(app.output_label(i), class_correct[i] * 100 / total_correct[i]))
```

Таблица 3. Отчет о классификации

Классы	Название класса	precision	recall	f1-score	support
0	T-shirt/Top	0.84	0.84	0.84	2570
1	Trouser	0.98	0.99	0.98	2462
2	Pullover	0.86	0.83	0.85	2407
3	Dress	0.91	0.91	0.91	2470
4	Coat	0.82	0.88	0.85	2490
5	Sandal	0.97	0.97	0.97	2533
6	Shirt	0.75	0.71	0.73	2513
7	Sneaker	0.94	0.94	0.94	2479
8	Bag	0.97	0.98	0.98	2540
9	Ankle Boot	0.95	0.96	0.96	2536
accuracy				0.90	25000
macro avg		0.90	0.90	0.90	25000
weighted avg		0.90	0.90	0.90	25000

precision (точность) - соотношение правильных положительных прогнозов к общему количеству положительных прогнозов.

recall (отзыв) - такой же как precision но к общему количеству фактических положительных результатов.

f1-score (f1-оценка) - оценка точность модели, средневзвешенное гармоническое значение.

Support (поддержка) — Количество классов.

Accuracy (точность) — оценка сходство правильных классов.

macro avg (Макросреднее) - среднее значение precision, recall, f1-score по всем классам.

weighted avg (Средневзвешенное значение) - среднее значение precision, recall, f1-score по взвешенное количеству классов.

Модель лучшее всего определяет классы — Trouser (1) и Bag (8) — 0,98, а худшее Shirt (6) — 0,73 по метрике f1-score.

Листинг 2.6. Формирования отчет классификация.

```
predictions_l = [df_m["predictions"].tolist()[i] for i in range(len(df_m["predictions"]))]
labels_l = [df_m["labels"].tolist()[i] for i in range(len(df_m["labels"]))]
predictions_l = list(chain.from_iterable(predictions_l))
labels_l = list(chain.from_iterable(labels_l))
confusion_matrix(labels_l, predictions_l)
print("Classification report for CNN :\n%s\n"
        % (metrics.classification_report(labels_l, predictions_l)))
```

3 Выводы

В данное статье была использована модель для классификация изображения одежды набора данных MNIST, в результате работы получены оценка качество модели и точность определение класса изображения, модель лучшее определяет классы классы — Trouser (1) и Bag (8) — 0,98, а худшее всего модель справился с Shirt (6) — 0,73 по метрике f1-score.

Библиографический список

1. Shoham N. et al. Overcoming forgetting in federated learning on non-iid data //arXiv preprint arXiv:1910.07796. – 2019.
2. Zheng W. et al. Target-based resource allocation for deep learning applications in a multi-tenancy system //2019 IEEE High Performance Extreme Computing Conference (HPEC). IEEE, 2019. С. 1-7.
3. Li Y., Liu H. Implementation of stochastic quasi-newton's method in pytorch //arXiv preprint arXiv:1805.02338. 2018.
4. Xiao H., Rasul K., Vollgraf R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms //arXiv preprint arXiv:1708.07747. 2017.
5. Yousefpour A. et al. Opacus: User-friendly differential privacy library in PyTorch //arXiv preprint arXiv:2109.12298. 2021.
6. McKenna M. A comparison of activation functions for deep learning on Fashion-MNIST //arXiv preprint arXiv:1708.07747. 2017.
7. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges // mnist URL: http://yann.lecun.com/exdb/mnist/ (дата обращения:

2023-05-22).

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
import torchvision.transforms.functional as F
from torch.autograd import Variable
from torch.utils.data import Dataset
from typing import Optional, Callable, TypeVar, Type, Union
from PIL import Image
from Lib.AppMain import *
import matplotlib.pyplot as plt
import os
from sklearn.metrics import confusion_matrix
import sklearn.metrics as metrics
from itertools import chain

class FashionDataset(Dataset):
	def __init__(self, data, transform = None):
		self.fashion_MNIST = list(data.values)
		self.transform = transform
		label = []
		image = []
		for i in self.fashion_MNIST:
			label.append(i[0])
			image.append(i[1:])
		self.labels = np.asarray(label)
		self.images = np.asarray(image).reshape(-1, 28, 28, 1).astype('float32')
	def __getitem__(self, index):
		label = self.labels[index]
		image = self.images[index]
		if self.transform is not None:
			image = self.transform(image)
		return image, label
	def __len__(self):
		return len(self.images)

class FashionCNN(nn.Module):
	def __init__(self):
		super(FashionCNN, self).__init__()
		self.layer1 = nn.Sequential(
			nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1),
			nn.BatchNorm2d(32),
			nn.ReLU(),
			nn.MaxPool2d(kernel_size=2, stride=2)
		)
		self.layer2 = nn.Sequential(
			nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3),
			nn.BatchNorm2d(64),
			nn.ReLU(),
			nn.MaxPool2d(2)
		)
		self.fc1 = nn.Linear(in_features=64*6*6, out_features=600)
		self.drop = nn.Dropout2d(0.25)
		self.fc2 = nn.Linear(in_features=600, out_features=120)
		self.fc3 = nn.Linear(in_features=120, out_features=10)
	def forward(self, x):
		out = self.layer1(x)
		out = self.layer2(out)
		out = out.view(out.size(0), -1)
		out = self.fc1(out)
		out = self.drop(out)
		out = self.fc2(out)
		out = self.fc3(out)
		return out

class App(AppMain):
	def output_label(self, label):
```

```
                output_mapping = {
                        0: "T-shirt/Top",
                        1: "Trouser",
                        2: "Pullover",
                        3: "Dress",
                        4: "Coat",
                        5: "Sandal",
                        6: "Shirt",
                        7: "Sneaker",
                        8: "Bag",
                        9: "Ankle Boot"
                }
                input = (label.item() if type(label) == torch.Tensor else label)
                return output_mapping[input]
        def main(self):
                self.dir_prefix = "./Data"
                self.dirs = {
                        "fig": "Fig",
                        "view": "View",
                        "view_test": "View_test"
                }
                self.profile_name = "default"
                self.datasets = deque()
                self.model = {}
                self.optimizer = {}
                self.auto_save_exit = False
                self.disp_metric = ["loss"]
                self.metric.loss = None
                self.metric.accuracy = None
                self.metric.predictions = None
                self.metric.labels = None
                self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
                self.init_dirs()
                self.criterion = nn.CrossEntropyLoss().to(self.device)
                self.lr = 0.001
                self.model = {
                        "FashionCNN": FashionCNN().to(self.device)
                }
                self.optimizer = {
                        "FashionCNN": torch.optim.Adam(self.model["FashionCNN"].parameters(), lr=self.lr)
                }

                self.transform = transforms.Compose([
                        transforms.ToTensor()
                ])
                self.init_data()
        def init_data(self):
                train_csv = pd.read_csv("fashion-mnist_train.csv")
                test_csv = pd.read_csv("fashion-mnist_test.csv")

                self.datasets.append(Datasets_batch(FashionDataset(train_csv, transform=self.transform), batch_size=100))
                self.datasets.append(Datasets_batch(FashionDataset(test_csv, transform=self.transform), batch_size=100))
                self.init_datasets(num_workers=0)
                self.cache_data = True
        def start_train(self) -> None:
                self.model["FashionCNN"].train()
        def step_train(self, sel: any, index :int) -> None:
                super().step_train(sel, index)
                model = self.model["FashionCNN"]
                optimizer = self.optimizer["FashionCNN"]
                images, labels = sel
                train = Variable(images.view(100, 1, 28, 28)).to(self.device)
                labels = Variable(labels).to(self.device)
                outputs = model(train)
                loss = self.criterion(outputs, labels)
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
                if not (self.metric.Step % 25):
                        total = 0
                        correct = 0
                        for images, labels in self.datasets[1].dataLoader:
                                images, labels = images.to(self.device), labels.to(self.device)

                                test = Variable(images.view(100, 1, 28, 28))

                                outputs = model(test)

```

```
                                        predictions = torch.max(outputs, 1)[1].to(self.device)
                                        correct += (predictions == labels).sum()

                                        total += len(labels)

                                accuracy = correct * 100 / total
                                self.metric.accuracy = accuracy
                                self.metric.predictions = predictions
                                self.metric.labels = labels
                        else:
                                self.metric.accuracy = None
                                self.metric.predictions = None
                                self.metric.labels = None

                        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 + "/" + app.output_label(int(labels[j])).split("/")[0]
                if not os.path.exists(cl):
                        os.mkdir(cl)
                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 + "/" + app.output_label(int(labels[j])).split("/")[0]
                if not os.path.exists(cl):
                        os.mkdir(cl)
                img_d = F.to_pil_image(images[j])
                app.save_image(cl + "/" + "%s.jpg" % i, img_d)
                i += 1

summary(app.model["FashionCNN"], input_size=(1, 28, 28), batch_size=100)

app.fit(10)
for i in range(len(app.metric._data)):
        if not app.metric._data[i]["accuracy"] is None:
                app.metric._data[i]["accuracy"] = float(app.metric._data[i]["accuracy"])
        if not app.metric._data[i]["predictions"] is None:
                app.metric._data[i]["predictions"] = app.metric._data[i]["predictions"].cpu().tolist()
        if not app.metric._data[i]["labels"] is None:
                app.metric._data[i]["labels"] = app.metric._data[i]["labels"].cpu().tolist()
app.save_model("FashionCNN")

#----------------------------
app.load_model("FashionCNN")

df = pd.DataFrame(app.metric._data)
df_m = df[df["accuracy"].notna()]
x_step = 600

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

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)

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


class_correct = [0. for _ in range(10)]
total_correct = [0. for _ in range(10)]

with torch.no_grad():
        for images, labels in app.datasets[1].dataLoader:
                images, labels = images.to(app.device), labels.to(app.device)
                test = Variable(images)
                outputs = app.model["FashionCNN"](test)
                predicted = torch.max(outputs, 1)[1]
                c = (predicted == labels).squeeze()

                for i in range(100):
                        label = labels[i]
                        class_correct[label] += c[i].item()
                        total_correct[label] += 1

for i in range(10):
        print("Accuracy of {}: {:.2f}%".format(app.output_label(i), class_correct[i] * 100 / total_correct[i]))

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

predictions_l = [df_m["predictions"].tolist()[i] for i in range(len(df_m["predictions"]))]
labels_l = [df_m["labels"].tolist()[i] for i in range(len(df_m["labels"]))]
predictions_l = list(chain.from_iterable(predictions_l))
labels_l = list(chain.from_iterable(labels_l))

confusion_matrix(labels_l, predictions_l)
print("Classification report for CNN :\n%s\n"
        % (metrics.classification_report(labels_l, predictions_l)))
```