

 <p>دانشگاه صنعتی خواجه نصیرالدین طوسی دانشکده مهندسی برق - کروه مهندسی کنترل</p>	<p>به نام خدا</p> <p>دانشگاه تهران - دانشگاه صنعتی خواجه نصیرالدین طوسی تهران</p> <p>دانشکده مهندسی برق و کامپیوتر</p>	
<h2>شبکه‌های رمزنگار - رمزگشای مولد</h2>		

محمد جواد احمدی	نام و نام خانوادگی
۴۰۱۰۰۰۸۶	شماره دانشجویی

فهرست مطالب

۳	پاسخ پرسش اول
۳	۱.۱ پاسخ قسمت ۱ - مجموعه دادگان مقاله
۴	۲.۱ پاسخ قسمت ۲ - انجام PCA و ISOMAP
۱۷	۳.۱ پاسخ قسمت ۳ - رمزگذار-رمزگشا
۲۷	۴.۱ پاسخ قسمت ۴ - خودرمزگذار متغیر (Variational AutoEncoder)
۳۹	۵.۱ پاسخ قسمت ۵ - کاوش در فضای latent
۴۳	۶.۱ پاسخ قسمت ۶ - Diffusion Models

فهرست تصاویر

۵	کاهش ابعاد داده‌ها با استفاده از روش تحلیل مولفه اساسی	۱
۵	تعامد بردارهای مولفه‌های اساسی اول و دوم به دلیل همبستگی صفر آن‌ها	۲
۶	نتایج تحلیل مولفه‌های اساسی روی تصویر	۳
۷	ایزوپ	۴
۱۳	نتایج ماتریس درهم‌ریختگی استفاده و عدم استفاده از PCA	۵
۱۷	نتایج ماتریس درهم‌ریختگی استفاده و عدم استفاده از ISOMAP	۶
۲۲	نمودار تابع اتلاف آموزش مدل رمزگذار-رمزگشای تمام‌امتصل	۷
۲۲	نتیجه طبقه‌بندی در حالت رمزگذار-رمزگشای تمام‌امتصل	۸
۲۷	نمودار تابع اتلاف آموزش مدل رمزگذار-رمزگشای کانولوشنی	۹
۲۷	نتیجه طبقه‌بندی در حالت رمزگذار-رمزگشای کانولوشنی	۱۰
۲۸	خودرمزگذار متغیر (Variational AutoEncoder)	۱۱
۳۴	نمودار تابع اتلاف آموزش مدل خودرمزگذار متغیر تمام‌امتصل	۱۲
۳۴	نتیجه طبقه‌بندی در حالت خودرمزگذار متغیر تمام‌امتصل	۱۳
۳۹	نمودار تابع اتلاف آموزش مدل خودرمزگذار متغیر کانولوشنی	۱۴
۳۹	نتیجه طبقه‌بندی در حالت خودرمزگذار متغیر کانولوشنی	۱۵
۴۴	کاوش در فضای نهان مدل خودرمزگذار متغیر تمام‌امتصل	۱۶
۴۵	کاوش در فضای نهان خودرمزگذار متغیر کانولوشنی	۱۷
۴۶	مقایسه مدل‌های مختلف شبکه‌های مولد	۱۸
۴۷	Diffusion Models	۱۹

پرسش ۱. شبکه‌های رمزگذار-رمزگشا مولد

۱ پاسخ پرسش اول

توضیح پوشۀ کدهای شبکه‌های رمزگذار-رمزگشا مولد

کدهای مربوط به این قسمت، علاوه بر پوشۀ محلی کدها در [این لینک گوگل کولب](#) آورده شده است. تصاویر و مدل‌های ذخیره شده هم از طریق [این لینک](#) در دسترس هستند.

۱.۱ پاسخ قسمت ۱ - مجموعه‌دادگان مقاله

با توجه به زوج‌بودن شماره دانشجوی (۴۰۱۰۰۸۶)، مجموعه‌داده Fashion MNIST را انتخاب کردام. برای اجرای اهداف مدنظر سوال ابتدا کتابخانه‌های لازم را فراخوانی می‌کنیم. سپس با دستور `fashion_mnist.load_data()` مجموعه‌داده MNIST را بارگیری می‌کنیم. داده‌های آموزش و آزمون به ترتیب در متغیرهای `y_train`, `x_train`, `y_test` و `x_test` ذخیره می‌شوند. در ادامه، `x_train` و `x_test` ماتریس‌های تصاویر ورودی هستند و `y_train` و `y_test` بردارهای برچسب‌های متاظر با تصاویر هستند. سپس مقادیر تصاویر را در محدوده ۰ تا ۱ نرمال می‌کنیم. ابتدا نوع داده تصاویر را به `float32` تغییر می‌دهیم و سپس تمام مقادیر را برابر ۲۵۵ تقسیم می‌کنیم. این عملیات باعث تبدیل مقادیر پیکسل‌ها به اعداد بین ۰ و ۱ می‌شود. در ادامه، تصاویر را به بردارهای یک‌بعدی تغییر شکل می‌دهیم. تصاویر ابتدا به شکلی مانند (تعداد نمونه‌ها، ابعاد تصویر) هستند و با استفاده از `[1:]` به شکل (`np.prod(x_train.shape[1:])`) تبدیل می‌شوند. این کار باعث می‌شود تمام ویژگی‌های تصاویر را در یک بردار یک بعدی نگه داریم. بردارهای برچسب `y_train` و `y_test` هم به شکل بردارهای یک‌بعدی با استفاده از `(1,-1)` `reshape` تغییر شکل می‌دهند. در انتهای، مقادیر داده‌های آموزش و آزمون استانداردسازی می‌شوند. ابتدا یک نمونه از `StandardScaler` ایجاد می‌شود. سپس با استفاده از `fit_transform`، پارامترهای استانداردسازی بر اساس داده‌های آموزش تعیین و داده‌های آموزش نرمال شده را به دست می‌آوریم. هم‌چنین با استفاده از `transform`، داده‌های آزمون را با استفاده از همان پارامترهای استانداردسازی آموزش، استانداردسازی می‌کنیم. دستورات مربوطه برای بارگیری و پیش‌پردازش داده‌ها در [برنامه ۱](#) آورده شده است.

Program 1: Load & Pre-process Data

```

1 import os
2 import numpy as np
3 import tensorflow as tf
4 from tensorflow.keras.datasets import fashion_mnist
5 from sklearn.preprocessing import StandardScaler
6
7 # Load Fashion MNIST dataset
8 (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
9

```

```

10 # Preprocessing
11 # Normalize values between 0 and 1
12 x_train = x_train.astype('float32') / 255.
13 x_test = x_test.astype('float32') / 255.
14
15 # Put images into vectors
16 x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
17 x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
18
19 # Reshape labels as vectors
20 y_train = y_train.reshape(-1)
21 y_test = y_test.reshape(-1)
22
23 # Standardize values (0 mean and unit variance)
24 scaler = StandardScaler()
25 x_train = scaler.fit_transform(x_train)
26 x_test = scaler.transform(x_test)

```

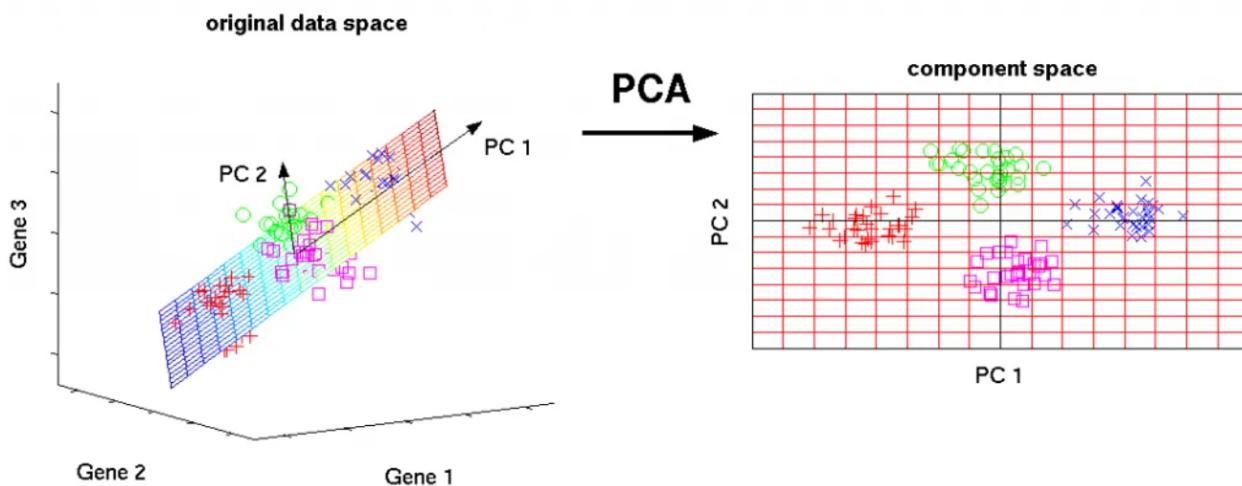
۲.۱ پاسخ قسمت ۲ - انجام PCA و ISOMAP

وقتی یک مجموعه داده متغیرها و ویژگی‌های زیادی داشته باشد، ممکن است بسیاری از این متغیرها و ویژگی‌ها با یکدیگر هم‌بستگی داشته باشند. هم‌چنین تحلیل تعداد زیادی از متغیر هزینه محاسباتی بالایی دارد. برای مثال اگر یک مجموعه داده ابعادی برابر با 50×49 در آن 300 تعداد کل نمونه‌ها و 50 تعداد متغیرهای پیش‌بینی است، به دلیل تعداد ابعاد زیاد (50)، می‌توان 50×49 نمودار پراندگی برای آن رسم کرد که این یعنی بیش از 1000 نمودار برای انجام تحلیل روی روابط بین متغیرها که هزینه محاسباتی و پیچیدگی بالایی دارد. تحلیل مولفه اساسی روشی برای استخراج متغیرهای مهم به صورت مولفه از مجموعه بزرگی از متغیرهای موجود در مجموعه داده است. تحلیل مولفه اساسی همیشه روی ماتریس کواریانس اعمال می‌شود و این یعنی باید داده‌ها عددی و استانداردسازی شوند. در این روش یک زیرمجموعه از متغیرهای پیش‌بینی که حاوی بیشترین اطلاعات درباره داده‌ها هستند برگزیده می‌شوند و این امکان فراهم می‌آید که نمودار پراکندگی داده‌ها در ابعاد پایین‌تری قابل ترسیم باشند. در [شکل ۱](#) نگاشت داده‌های دارای سه بعد به داده‌های با دو بعد با روش تحلیل مولفه اساسی کاهش بعد داده شده اند که در آن هر بعد در فضای جدید یک ترکیب خطی از ویژگی‌های اصلی است.

یک مولفه اساسی یک ترکیب خطی نرمال شده از ویژگی‌های اصلی موجود است (PC1 و PC2 در [شکل ۱](#)). اگر یک مجموعه از متغیرهای ویژگی و پیش‌بینی به صورت X^1, X^2, \dots, X^p در اختیار داریم که متغیرهایی نرمال شده با میانگین صفر و انحراف معیار یک هستند. مولفه‌های اساسی این مجموعه از متغیرها را می‌توان به صورت زیر نوشت:

$$Z^1 = \Phi^{11} X^1 + \Phi^{21} X^2 + \Phi^{31} X^3 + \dots + \Phi^{p1} X^p \quad (1)$$

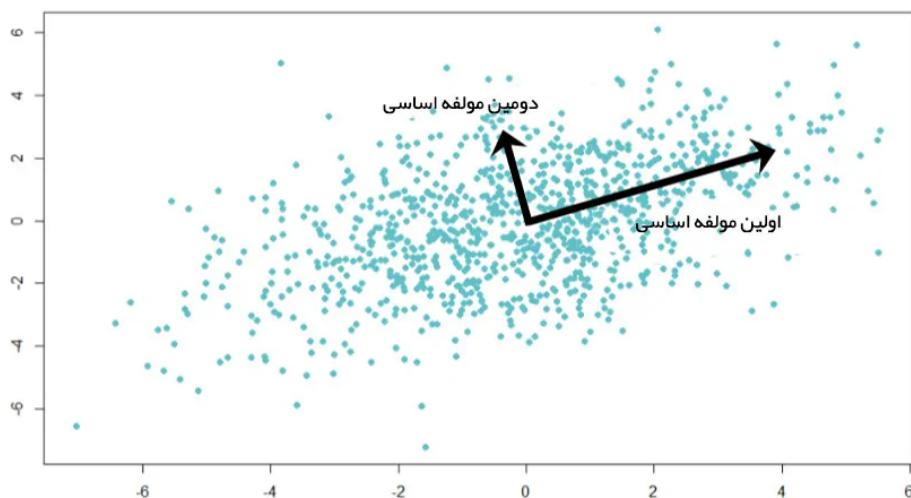
که در آن Z^1 اولین مولفه اساسی، Φ^{p1} بردار بار شامل بردارهای بار اولین مولفه اساسی است. بردارهای بار به مجموع مربعات مساوی با یک محدود شده‌اند تا مقادیر بار بزرگ منجر به ایجاد واریانس بسیار بزرگ نشود. این مقدار جهت مولفه اساسی را در جهتی که داده‌ها بیشترین تنوع را دارند تعریف می‌کند. اولین مولفه اساسی، یک ترکیب خطی از متغیرهای اصلی است که بیشترین واریانس موجود در مجموعه داده‌ها را در بر می‌گیرد. این مولفه، جهت بیشترین تغییرات در داده‌ها را تعیین می‌کند.



شکل ۱: کاهش ابعاد داده‌ها با استفاده از روش تحلیل مولفه اساسی.

هرچه دامنه تغییرات موجود در اولین مولفه بیشتر باشد، اطلاعات موجود در این مولفه بیشتر است. در نتیجه محاسبه این مولفه، خطی به دست می‌آید که نزدیکترین خط به داده‌ها محسوب می‌شود. در واقع این خط مجموع مربع فواصل را بین یک نقطه داده و خط، به کمینه مقدار می‌رساند. دومین مولفه اساسی (Z^2) هم یک ترکیب خطی از متغیرهای اصلی است که واریانس باقیمانده در مجموعه داده را در خود حفظ می‌کند و با مقدار Z^1 ناهمبسته است. به عبارت دیگر، همبستگی بین مولفه اساسی اول و دوم صفر است. اگر دو مولفه ناهمبسته باشند، جهت‌های آن‌ها باید متعامد باشند. این موضوع در شکل ۲ نشان داده شده است. مولفه اساسی دوم را می‌توان به شکل زیر نمایش داد:

$$Z^2 = \Phi^{12}X^1 + \Phi^{22}X^2 + \Phi^{32}X^3 + \dots + \Phi{p2}X_p \quad (2)$$

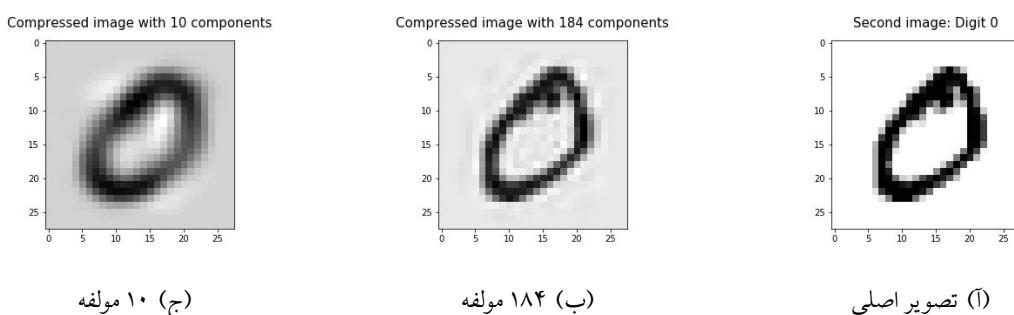


شکل ۲: تعامد بردارهای مولفه‌های اساسی اول و دوم به دلیل همبستگی صفر آن‌ها.

تمامی مولفه‌های اساسی بعدی نیز از مفهومی مشابه آنچه بیان شد، پیروی می‌کنند. به عبارت دیگر، آن‌ها مقدار واریانس

باقیمانده را بدون آنکه با مولفه‌های پیشین دارای همبستگی شوند، در خود حفظ می‌کنند. به طور کلی، در داده‌های دارای $n \times p$ بعد، به میزان $p - \min(n-1)$ مولفه اساسی قابل ایجاد است. لازم به ذکر است که تحلیل مولفه اساسی روی نسخه نرم‌المل شده متغیرهای اصلی اعمال می‌شود. این امر به آن دلیل است که متغیرهای اصلی ممکن است مقیاس‌های گوناگونی داشته باشند.

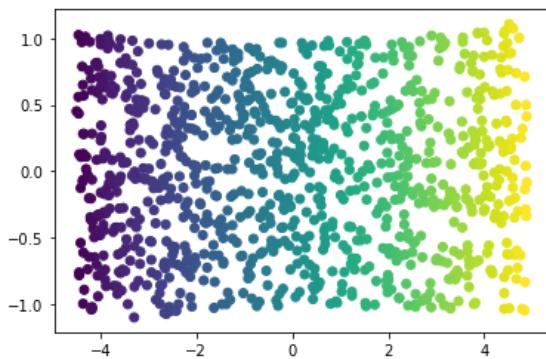
وقتی روش PCA را روی تصاویر با ابعاد ۲۸ × ۲۸ اعمال می‌کنیم، هدفمان این است که از مجموعه‌ای از تصاویر به عنوان ورودی، یک مجموعه کوچک‌تر و کم ابعادتر از ویژگی‌ها (یا اجزای اصلی) را استخراج کنیم که بیشترین اطلاعات را درباره تصاویر اصلی دارند. ابتدا، برای اعمال PCA روی تصاویر، باید داده‌ها را به فضای ویژگی تبدیل کنیم. در اینجا، هر تصویر با ابعاد ۲۸ × ۲۸ به صورت یک بردار با ۷۸۴ ابعاد در نظر گرفته می‌شود. سپس، برای محاسبه ماتریس کوواریانس بین ویژگی‌ها، نیاز به محاسبه میانگین و کاهش آن از داده‌ها است. مرحله بعدی در PCA، محاسبه بردارهای ویژه است. بردارهای ویژه نشان می‌دهند که در چه جهت‌هایی ماتریس کوواریانس بیشترین و کمترین واریانس را دارند. این بردارهای ویژه را می‌توان به عنوان محورهای جدید فضای ویژگی تصور کرد. با مرتب کردن بردارهای ویژه بر اساس مقادیر ویژه متناظر، می‌توانیم بردارهای ویژه مهم‌تر را انتخاب کنیم و تعدادی از بردارهای ویژه که مقادیر ویژه بزرگتری دارند را به عنوان اجزای اصلی انتخاب کنیم. در نهایت، با استفاده از اجزای اصلی انتخاب شده، می‌توانیم تصاویر اصلی را با دقت خوبی بازسازی کنیم. همچنین، اجزای اصلی معمولاً مرتبط با ویژگی‌های مهم تصاویر هستند، بنابراین می‌توانند به عنوان نماینده‌ی مهم‌ترین اطلاعات تصاویر در نظر گرفته شوند. بنابراین، تصاویر می‌توانند با اتصال پیکسل‌های تصویر ستون به ستون (یا به طور جایگزین سطر به سطر) به عنوان یک بردار یک بعدی بسیار بلند در نظر گرفته شوند. از آنجایی که جهت‌های PCA بسیار حساس به مقیاس داده هستند، قبل از اعمال آن باید مقیاس ویژگی‌ها را تغییر دهیم. ارزش‌های پیکسل هر تصویر از ۰ تا ۲۵۵ (مقیاس مشابه) متغیر است. ابتدا، باید تعداد مناسبی از ابعاد (به عبارتی، تعداد مناسبی از مولفه‌های اصلی) را انتخاب کنیم. برای این کار، PCA را با تعداد ابعاد اصلی (به عبارتی، برای تصاویر ۲۸ × ۲۸) اعمال کرده و نمودار را ایجاد می‌کنیم تا بینیم که PCA چه قدر واریانس داده اصلی را حفظ می‌کند. اگر از ۱۰ اجزای اول برای فشرده‌سازی تصویر استفاده کنیم، این اجزا تنها بخش کمی از تنوع موجود در داده اصلی را حفظ می‌کنند. بنابراین، ممکن است تصویر واضحی دریافت نکنیم. اگر این تصویر حاصله را با تصویر اصلی که قبل از آورده‌ایم مقایسه کنیم، بسیار واضح نیست و از اطلاعات کافی برخوردار نیست. اما اگر مثلاً از ۱۸۴ اجزای اول برای فشرده‌سازی تصویر استفاده کنیم، مشاهده می‌کنیم که درصد بالایی از تغییرات داده اصلی توسط این ۱۸۴ مولفه حفظ می‌شود. بنابراین، در این بار، یک تصویر بسیار واضح که بسیار شبیه به تصویر اصلی است دریافت خواهیم کرد. نمونه‌ای از این فرآیند در **شکل ۳** نشان داده شده است.



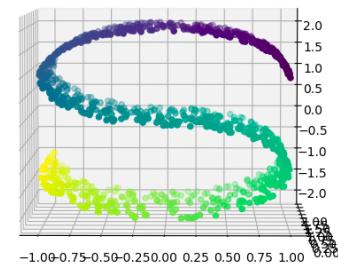
شکل ۳: نتایج تحلیل مولفه‌های اساسی روی تصویر.

روش ISOMAP یک روش غیرخطی است که در آن از شیوه geodesic distance استفاده می‌شود. برای نقاط نزدیک، فاصله اقلیدسی به عنوان تقریب مورد استفاده قرار می‌گیرد ($\epsilon \leq \|x^r - x^s\|$) و برای نقاط دورتر مجموع فواصل نقاط نزدیک مورد استفاده قرار می‌گیرد. با تشکیل گراف فواصل، می‌توانیم تقریبی از فاصله نقاط روی یک خمینه به دست بیاوریم. بعد از

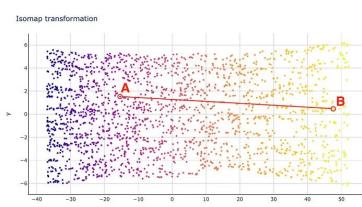
محاسبه فواصل می‌توان از MDS برای کاهش ابعاد استفاده کرد. با افزایش نقاط، ضمن افزایش دقت، پیچیدگی محاسباتی هم زیاد می‌شود. به صورت کلی ایزومپ نقشه‌برداری ایزومنتریک است. هدف این نقشه‌برداری حفظ فاصله ژئودزیک بین دو نقطه است. ژئودزیک به طور رسمی به عنوان کوتاه‌ترین مسیر بر روی خود سطح تعریف می‌شود. با درک فواصل ژئودزیک دو به دو، هدف ایزومپ تخمین هندسه داده‌ها قبل از طرح ریزی آن در ابعاد مشخص است. ایزومپ در ۳ مرحله اصلی عمل می‌کند: ابتدا یک نمودار همسایگی با روش‌هایی مانند kNN ساخته می‌شود. سپس کوتاه‌ترین مسیر بین نقاط را از طریق الگوریتم هایی مانند Dijkstra و فلوید-وارشال محاسبه می‌کنیم. درنهایت، یک دسته‌بندی d بعدی با تجزیه مقدار ویژه جزئی (یعنی گرفتن بزرگ‌ترین مقادیر ویژه کرنل) می‌سازیم. به عنوان یک مثال می‌توانیم با استفاده از `sklearn.datasets`، ۱۰۰۰ نقطه روی `-curves`-تولید کنیم. سپس تلاش می‌کنیم تا با استفاده از ایزومپ آن را به دو بعد کاهش دهیم. درست مانند kNN، تعداد همسایگان پارامتری است که باید برای عملکرد ایده‌آل اصلاح شود. اگر از عدد ۳۵ استفاده کنیم همانطور که در [شکل ۴](#) می‌بینیم، بخش زرد دورترین بخش از بنفس تیره / آبی در طرح سه‌بعدی است، که نشان می‌دهد که می‌خواهیم آن بخش‌های manifold از یکدیگر دورتر باشند، که چیزی است که در طول کوتاه‌ترین مسیر ساخت به دست می‌آید.



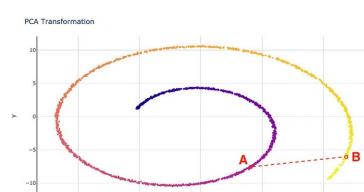
(ب) کاهش به دو بعد



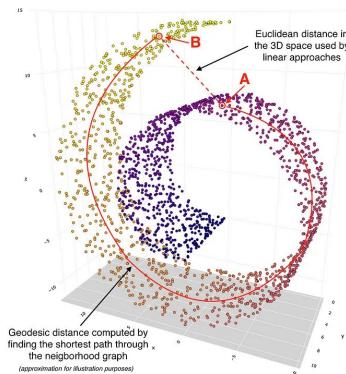
s-curve (۱)



ISOMAP (۵)



PCA (۶)



s-curve (۷)

شکل ۴: ایزومپ.

این الگوریتم مناسب برای تحلیل ترتیب نزدیکی (Manifold Learning) است که برای تبدیل داده‌های برداری از فضای بعد بالا به فضای بعد کمتر استفاده می‌شود. هدف اصلی آن، حفظ نزدیکی‌ها در مجموعه داده‌ها است، به طوری که فاصله‌های اقلیدسی بین داده‌ها در فضای کاهش یافته حفظ شود. برای این اهداف در ابتدا، برای هر نمونه داده در مجموعه داده‌ها، فاصله‌های

اقلیدسی بین آن نمونه و سایر نمونه‌ها را محاسبه می‌کنیم. این محاسبات می‌تواند با استفاده از فاصله اقلیدسی ساده یا از روش‌های دیگری مانند فاصله منهتن (Manhattan distance) صورت بگیرد. نتیجه این مرحله یک ماتریس فاصله است که نمایانگر فاصله بین هر دو نمونه داده است. با استفاده از ماتریس فاصله، یک گراف مشابه ساخته می‌شود. هر نمونه داده به عنوان یک گره در گراف در نظر گرفته می‌شود و یال‌ها بین نمونه‌ها بر اساس فاصله‌های محاسبه شده قرار می‌گیرند. در این گراف، هر نمونه داده با همسایگان نزدیک خود متصل می‌شود و فاصله بین همسایگان نزدیک بر روی یال‌ها قرار می‌گیرد. با استفاده از الگوریتم کوتاه‌ترین مسیرها (Shortest Path Algorithm) مانند الگوریتم Dijkstra، ماتریس کوتاه‌ترین مسیرها بین هر دو نمونه در گراف محاسبه می‌شود. این ماتریس نشان می‌دهد که هر نمونه داده چه فاصله‌ای از هر نمونه داده دیگری دارد. با استفاده از روش کاهش بعد مانند تجزیه مقادیر ویژه، ماتریس کوتاه‌ترین مسیرها کاهش بعد می‌یابد. برای این کار، ابتدا ماتریس کوتاه‌ترین مسیرها را به یک ماتریس شباهت تبدیل می‌کنیم که مشابه به ماتریس فاصله اولیه است. سپس با استفاده از روش‌های تجزیه مقادیر ویژه بر روی این ماتریس شباهت، بردارهای ویژه متناظر با مقادیر ویژه بزرگتر را استخراج می‌کنیم. این بردارهای ویژه، نمایش کاهش یافته‌ای از مجموعه داده‌ها در فضای کاهش یافته هستند. با استفاده از بردارهای ویژه به دست آمده، می‌توانیم نمونه‌های جدید را در فضای کاهش یافته بازسازی کنیم. بازسازی فضای کاهش یافته بر اساس محاسبه ترکیب خطی از بردارهای ویژه بر روی بردارهای ورودی انجام می‌شود.

برای اعمال ایرومپ روی تصویر ابتدا تصاویر با ابعاد ۲۸ در ۲۸ را به عنوان نقاط در فضای ۷۸۴-بعدی تعریف می‌کنیم. سپس برای هر زوج نقاط، ماتریس فاصله را محاسبه می‌کنیم. می‌توانیم از فاصله اقلیدسی استفاده کنیم که معمولاً برای تصاویر استفاده می‌شود. با استفاده از ماتریس فاصله، نمودار همسایگی را ساخته و روابط همسایگی بین نقاط را بررسی می‌کنیم. برای هر نقطه، نقاطی که نزدیکترین همسایه‌های آن‌ها هستند را به عنوان همسایگان آن نقطه در نظر می‌گیریم. با استفاده از نمودار همسایگی، ماتریس دسترسی را محاسبه می‌کنیم. این ماتریس شامل کوتاه‌ترین مسیرهای ممکن بین هر زوج نقاط است. معمولاً از الگوریتم دیجکسترا (Dijkstra) برای محاسبه ماتریس دسترسی استفاده می‌شود. با استفاده از ماتریس دسترسی، نقاط را در یک فضای کمتر ابعاد قرار می‌دهیم. برای این کار می‌توان از روش‌های کاهش ابعاد مانند تجزیه مقادیر ویژه و یا تحلیل مؤلفه‌های اصلی استفاده کرد.

به عنوان جمع‌بندی، PCA اجازه می‌دهد تا اطلاعات به صورت خطی ترکیب شوند و به این ترتیب، اطلاعات غیرضروری را حذف کرده و اهمیت بالایی به اطلاعات مهم می‌دهد. این روش هم‌چنین ناحیه بین‌نهایت متغیری از داده‌ها را کاهش می‌دهد. این به این معنی است که برای هر نوع دسته‌بندی یا کاربردی قابل استفاده است. PCA محاسبات ساده‌تری نسبت به برخی از روش‌های کاهش بعد داده‌ها دارد. این روش بر اساس مقادیر و بردارهای ویژه ماتریس کواریانس داده‌ها عمل می‌کند و به راحتی قابل پیاده‌سازی است. این روش هم‌چنین به خوبی در کاربردهای مختلفی مانند تصویربرداری، تحلیل داده‌های مالی، تشخیص الگو و فشرده‌سازی اطلاعات مورد استفاده قرار می‌گیرد. PCA توانایی تفسیر پذیری هم دارد، به این معنا که ویژگی‌هایی که در اثر تجزیه و تحلیل از طریق PCA استخراج می‌شوند، با محاسبه بردارهای ویژه مربوط به هر بعد، قابل توجیه و تفسیر هستند. با وجود این مزایا، PCA تنها برای روابط خطی مناسب است و برای روابط غیرخطی به صورت کارآمد عمل نمی‌کند. در برخی موارد، کاهش ابعاد باعث از دست رفتن بخشی از اطلاعات می‌شود و ممکن است باعث افت کیفیت داده‌ها شود. PCA به مقیاس داده‌ها حساس است. مقادیر متغیرهای با مقیاس‌های بزرگتر بیشترین تأثیر را در تحلیل خروجی دارند. از طرف دیگر، ISOMAP به خوبی روابط غیرخطی را در داده‌ها حفظ می‌کند. این روش بر پایه مقاومت به تغییرات جابجایی است و از شکل جابجایی داده‌ها در فضای بعد بالاتر استفاده می‌کند. این روش هم‌چنین به خوبی فاصله‌ها را در داده‌ها حفظ می‌کند. این روش مقاومت زیادی در برابر نویز دارد و توانایی دارد تا اطلاعات مهم را در میان نویزها استخراج کند. این روش بر اساس نزدیکی‌ها و فواصل هندسی بین نقاط داده‌ها عمل می‌کند و تلاش می‌کند نقاط مجاور را به خوبی حفظ کند و به روشنی که نقاط متناظر در فضای بعد کم بعد حفظ شوند، تبدیل می‌شود. ISOMAP می‌تواند الگوهای غیرخطی را به خوبی شناسایی کند. در صورت

وجود نویز در داده‌ها و یا تبدیلات غیرخطی، ISOMAP می‌تواند عملکرد بهتری نسبت به PCA داشته باشد. ایزوپ هم‌چنین قادر است ساختار مکانی داده‌ها را در فضای بعد کم بعد حفظ کند. این به ما کمک می‌کند تا الگوها و ارتباطات پنهان در داده‌ها را شناسایی کنیم. با وجود این مزایا، ایزوپ هزینه محاسباتی بالایی دارد. محاسبه ماتریس فاصله گراف نیاز به محاسبه مسیرهای کوتاهترین در گراف دارد که ممکن است زمان براش. این محاسبات می‌توانند منجر به پیچیدگی محاسباتی بالا در مورد مجموعه‌های داده بزرگ شوند. ایزوپ هم‌چنین نیاز به وجود نزدیک‌ترین همسایگان در فضای ورودی دارد تا بتواند روابط فضایی را تعیین کند. در بعضی موارد، این وابستگی به همسایگان ممکن است با مشکلاتی روبرو شود، مانند توزیع نامتناسب داده‌ها. ایزوپ به پارامترهایی مانند تعداد همسایه‌ها و یا شعاع نزدیک‌ترین همسایه‌ها هم حساس است. تعیین بهینه‌ی این پارامترها ممکن است چالشی باشد. در کل، PCA برای کاهش ابعاد سریع و ساده‌تر و در مواردی که خطی بودن الگوها معقول استفاده می‌شود، در حالی که ISOMAP برای تشخیص الگوهای غیرخطی و حفظ نزدیکی‌ها مناسب است. انتخاب روش مناسب بستگی به ماهیت داده‌ها و هدف تحلیل دارد.

برای پیاده‌سازی روش‌ها مطابق خواسته صورت سوال دستوراتی نوشته‌ایم. ایندا روش PCA را پیاده‌سازی می‌کنیم. در دستورات این بخش ابتدا کتابخانه‌های ضروری را فراخوانی می‌کنیم. سپس، مجموعه‌داده Fashion MNIST را بارگیری کرده و داده‌ها و برچسب‌ها را در متغیرهای X و y قرار می‌دهیم. در ادامه مطابق توضیحات و دستوراتی که در [بخش ۱.۱](#) ارائه شد داده‌ها را پیش‌پردازش می‌کنیم. ویژگی‌ها مقیاس‌بندی می‌شوند تا مقادیر آنها در بازه‌ای مشخص قرار بگیرند و تاثیر واحدهای مختلف مقیاس بر روی مدل را کاهش دهند. در ادامه، مجموعه‌داده به دو قسمت تقسیم می‌شود: داده‌های آموزش و داده‌های آزمون. ۸۰٪ درصد داده‌ها به عنوان داده‌های آموزش استفاده می‌شوند و ۲۰٪ درصد دیگر به عنوان داده‌های آزمون مورد استفاده قرار می‌گیرند. این تقسیم به وسیله تابع `train_test_split` صورت می‌گیرد. سپس، ابعاد داده با استفاده از PCA کاهش می‌یابند. ابعاد داده با استفاده از یک جستجوی تصادفی برای پیدا کردن تعداد مناسبی از مولفه‌ها پیاده‌سازی می‌شود. ایندا یک شیء PCA ایجاد می‌شود و سپس با استفاده از تابع `fit_transform` ابعاد داده آموزش کاهش داده و در $X_{\text{train}}.pca$ قرار می‌گیرند. همچنین، ابعاد داده آزمون نیز با استفاده از این تابع کاهش داده می‌شود و در $X_{\text{test}}.pca$ قرار می‌گیرند. در ادامه دو شیء از مدل دسته‌بندی KNN (با و بدون استفاده از PCA) تعریف می‌شوند. پارامترهای مورد نیاز برای جستجوی تصادفی برای مدل KNN هم تعریف می‌شوند. پارامترهای مورد نظر شامل $n_{\text{neighbors}}$ که تعداد همسایگان نزدیک در الگوریتم است و $weights$ که نحوه وزن دهی به همسایگان نزدیک را تعیین می‌کند، می‌باشند. در دستورات مربوط به جستجوی تصادفی پس از تعریف شیء مدنظر، یک `param_grid` تعریف می‌شود که حاوی تنها یک پارامتر است که نام آن $n_{\text{components}}$ است. مقادیر ممکن برای این پارامتر با استفاده از تابع `randint` از مارژول `scipy.stats` تعیین می‌شود. ($[1]$)، یک عدد به این معنی است که از بین اعداد صحیح بین ۱ تا تعداد اجزای موجود در داده‌های آموزش ($[1]$)، یک عدد تصادفی انتخاب می‌شود. در `nhlil`، یک شیء از کلاس `RandomizedSearchCV` برای جستجوی تصادفی ساخته می‌شود. این شیء به عنوان تخمین‌گر `pca` بوده و دامنه پارامترها (`param_grid_pca`) را دریافت می‌کند. همچنین، تعداد تکرار جستجو (`n_iter`) برابر با ۱۰ و تقسیم‌بندی متقاطع (`cv`) برابر با ۳ تعیین می‌شوند. در انتهای، جستجوی تصادفی با استفاده از روش PCA روی داده‌های آموزش (X_{train}) انجام می‌شود. این جستجو با فراخوانی تابع `fit` بر روی شیء `random_search_pca` انجام می‌شود. درنهایت ضمن نمایش بهترین پارامترها، طبقه‌بندی انجام شده و نتایج در دو حالت استفاده از کاهش بعد و عدم استفاده از کاهش بعد مقایسه می‌شوند. دستورات در [برنامه ۲](#) آورده شده است.

Program 2: PCA + kNN Implementation

```

1 import time
2 import numpy as np
3 import matplotlib.pyplot as plt

```

```

4 from sklearn.datasets import fetch_openml
5 from sklearn.model_selection import train_test_split, RandomizedSearchCV
6 from sklearn.decomposition import PCA
7 from sklearn.neighbors import KNeighborsClassifier
8 from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
9 from sklearn.preprocessing import StandardScaler
10 from scipy.stats import randint
11 import warnings
12 warnings.filterwarnings('ignore')
13
14 # Load Fashion MNIST dataset
15 fashion_mnist = fetch_openml(name='Fashion-MNIST', version=1)
16 X = fashion_mnist.data
17 y = fashion_mnist.target
18
19 # (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
20
21 # # Preprocess the data
22 # x_train = x_train.reshape(-1, 784) / 255.0
23 # x_test = x_test.reshape(-1, 784) / 255.0
24
25 # Preprocessing: Scale the features
26 scaler = StandardScaler()
27 X_scaled = scaler.fit_transform(X)
28
29 # Split the dataset into training and test sets
30 X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
31
32 # Perform dimensionality reduction using PCA with random search for number of components
33 pca = PCA()
34 param_grid_pca = {'n_components': randint(1, X_train.shape[1])}
35 random_search_pca = RandomizedSearchCV(pca, param_distributions=param_grid_pca, n_iter=10, cv=3)
36 random_search_pca.fit(X_train)
37
38 # # Perform dimensionality reduction using PCA with random search for number of components
39 # pca = PCA()
40 # param_grid_pca = {'n_components': np.arange(int(X_train.shape[1] * 0.95), X_train.shape[1] + 1)}
41 # random_search_pca = RandomizedSearchCV(pca, param_distributions=param_grid_pca, n_iter=10, cv
42 #                                         =3)
43 # random_search_pca.fit(X_train)
44
45 # Get the best number of components and best parameters for PCA
46 best_n_components = random_search_pca.best_params_['n_components']
best_params_pca = random_search_pca.best_params_

```

```

47
48 # Apply PCA with the best number of components
49 pca = PCA(n_components=best_n_components)
50 X_train_pca = pca.fit_transform(X_train)
51 X_test_pca = pca.transform(X_test)
52
53 # Define the KNN classifiers
54 knn_without_pca = KNeighborsClassifier()
55 knn_with_pca = KNeighborsClassifier()
56
57 # Define the parameter grid for random search
58 param_grid_knn = {
59     'n_neighbors': randint(1, 20),
60     'weights': ['uniform', 'distance']
61 }
62
63 # Perform random search for parameter tuning without PCA
64 random_search_without_pca = RandomizedSearchCV(knn_without_pca, param_distributions=
65     param_grid_knn, n_iter=10, cv=3)
66 start_time_without_pca = time.time()
67 random_search_without_pca.fit(X_train, y_train)
68 end_time_without_pca = time.time()
69
70 # Perform random search for parameter tuning with PCA
71 random_search_with_pca = RandomizedSearchCV(knn_with_pca, param_distributions=param_grid_knn,
72     n_iter=10, cv=3)
73 start_time_with_pca = time.time()
74 random_search_with_pca.fit(X_train_pca, y_train)
75 end_time_with_pca = time.time()
76
77 # Predict labels for the test set without PCA
78 start_time_pred_without_pca = time.time()
79 y_pred_without_pca = random_search_without_pca.predict(X_test)
80 end_time_pred_without_pca = time.time()
81
82 # Predict labels for the test set with PCA
83 start_time_pred_with_pca = time.time()
84 y_pred_with_pca = random_search_with_pca.predict(X_test_pca)
85 end_time_pred_with_pca = time.time()
86
87 # Compute the accuracy without PCA
88 accuracy_without_pca = accuracy_score(y_test, y_pred_without_pca)
89 accuracy_with_pca = accuracy_score(y_test, y_pred_with_pca)

```

```

90
91 # Compute the confusion matrix without PCA
92 cm_without_pca = confusion_matrix(y_test, y_pred_without_pca)
93
94 # Compute the confusion matrix with PCA
95 cm_with_pca = confusion_matrix(y_test, y_pred_with_pca)
96
97 # Calculate the percentage values for the confusion matrices
98 cm_without_pca_percent = cm_without_pca / cm_without_pca.sum(axis=1)[:, np.newaxis]
99 cm_with_pca_percent = cm_with_pca / cm_with_pca.sum(axis=1)[:, np.newaxis]
100
101 # Print all results
102 print("Results without PCA:")
103 print("Best Parameters: ", random_search_without_pca.best_params_)
104 print("Accuracy: ", accuracy_without_pca)
105 print("Classification Time: ", end_time_without_pca - start_time_without_pca)
106 print("Prediction Time: ", end_time_pred_without_pca - start_time_pred_without_pca)
107 print()
108 print("Results with PCA:")
109 print("Best Parameters: ", random_search_with_pca.best_params_)
110 print("Accuracy: ", accuracy_with_pca)
111 print("Classification Time: ", end_time_with_pca - start_time_with_pca)
112 print("Prediction Time: ", end_time_pred_with_pca - start_time_pred_with_pca)
113
114 # Print the best components and parameters of PCA
115 print("Best Components (PCA):", best_n_components)
116 print("Best Parameters (PCA):", best_params_pca)

```

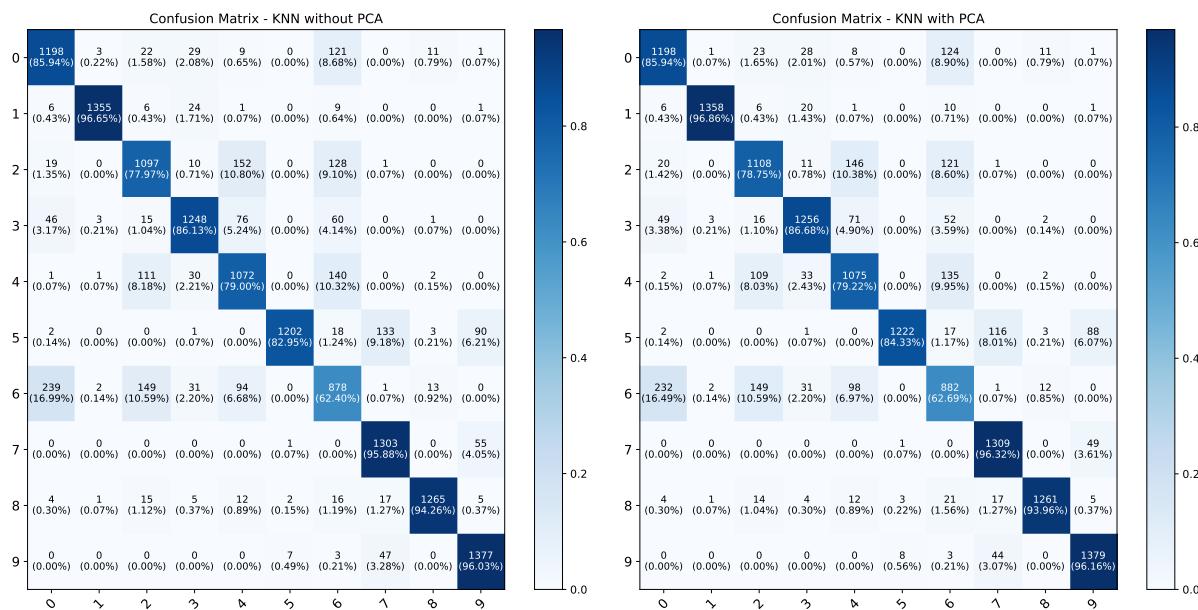
نتایج در [برنامه ۳](#) و [شکل ۵](#) آورده شده است. همان‌طور که مشاهده می‌شود حدود شش درصد از ابعاد کاهش یافته و نتایج قابل قبول است. هم‌چنین زمان اجرا هم در حالت کاهش بعد یافته کمتر است. هم‌چنین اختلاف بهترین پارامترهای kNN در حالت بدون استفاده از PCA و با استفاده از آن خروجی گرفته شده است.

Program 3: PCA + kNN Results

```

1 Results without PCA:
2 Best Parameters: {'n_neighbors': 7, 'weights': 'distance'}
3 Accuracy: 0.8567857142857143
4 Classification Time: 1279.7628347873688
5 Prediction Time: 47.87683057785034
6
7 Results with PCA:
8 Best Parameters: {'n_neighbors': 6, 'weights': 'distance'}
9 Accuracy: 0.8605714285714285
10 Classification Time: 1206.4444375038147
11 Prediction Time: 44.352290630340576
12 Best Components (PCA): 735
13 Best Parameters (PCA): {'n_components': 735}

```



شکل ۵: نتایج ماتریس درهم‌ریختگی استفاده و عدم استفاده از PCA.

مشابه PCA دستوراتی را هم برای پیاده‌سازی بخش ISOMAP می‌نویسیم. در اینجا هم ابتدا کتابخانه‌های ضروری را فراخوانی کرده و داده‌ها را بارگیری می‌کنیم و پیش‌پردازش‌های لازم را اعمال می‌کنیم. سپس، یک زیرمجموعه از مجموعه داده ایجاد می‌کنیم تا در صورت ایجاد مشکلات پیچیدگی محاسباتی از آن استفاده کنیم. سپس مجموعه داده را به دو بخش آموزش و آزمون تقسیم می‌کنیم. در اینجا ۸۰ درصد داده برای آموزش و ۲۰ درصد برای آزمون استفاده می‌شود. سپس تابع `evaluate_knn` را تعریف می‌کنیم که با استفاده از الگوریتم kNN، عملکرد دسته‌بندی را ارزیابی می‌کند. در مرحله بعد، عملکرد kNN بدون استفاده از روش ISOMAP را ارزیابی می‌کنیم. در مرحله بعد، عملکرد kNN با استفاده از روش ISOMAP را ارزیابی می‌کنیم. در اینجا از جستجوی تصادفی برای انتخاب بهترین پارامترها استفاده می‌شود. درنهایت نتایج را نمایش داده و مقایسه می‌کنیم و داده‌های غیرضروری را برای آزادشدن فضای حذف می‌کنیم. دستورات در برنامه ۴ آورده شده است و نتایج در [بنامه ۵](#) و [شکل ۶](#) نشان داده شده است. همان‌طور که مشاهده می‌شود، با کاهش ابعاد زمان اجرای طبقه‌بندی کاهش پیدا کرده است و نتایج هم قابل قبول بوده است.

Program 4: ISOMAP + kNN Implementation

```

1 import numpy as np
2 import time
3 from sklearn.datasets import fetch_openml
4 from sklearn.decomposition import KernelPCA
5 from sklearn.manifold import Isomap
6 from sklearn.model_selection import RandomizedSearchCV, train_test_split
7 from sklearn.neighbors import KNeighborsClassifier
8 from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
9 import matplotlib.pyplot as plt
10 import seaborn as sns
11 import warnings

```

```
12
13 warnings.filterwarnings('ignore')
14
15 # Load Fashion MNIST dataset (use a smaller subset)
16 X, y = fetch_openml('Fashion-MNIST', version=1, return_X_y=True, as_frame=False)
17 # subset_indices = np.random.choice(len(X), size=10000, replace=False)
18 # X = X[subset_indices]
19 # y = y[subset_indices]
20
21 # (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
22
23 # # Preprocess the data
24 # x_train = x_train.reshape(-1, 784) / 255.0
25 # x_test = x_test.reshape(-1, 784) / 255.0
26
27 # Preprocess data
28 X = X / 255.0
29
30 # Split the dataset into train and test sets
31 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
32
33 # Function to evaluate KNN classifier
34 def evaluate_knn(X_train, y_train, X_test, y_test):
35     start_time = time.time()
36     knn = KNeighborsClassifier(n_neighbors=5)
37     knn.fit(X_train, y_train)
38     y_pred = knn.predict(X_test)
39     end_time = time.time()
40     execution_time = end_time - start_time
41     print("KNN Classification Report:")
42     print(classification_report(y_test, y_pred))
43     print("KNN Execution Time: {:.2f} seconds".format(execution_time))
44     print("\n")
45     return y_pred, execution_time
46
47 # Without ISOMAP
48 print("Without ISOMAP:")
49 y_pred_no_isomap, execution_time_no_isomap = evaluate_knn(X_train, y_train, X_test, y_test)
50
51 # With ISOMAP
52 print("With ISOMAP:")
53 n_components = np.arange(10, 31, 10) # Reduce the range of components
54 n_neighbors = np.arange(5, 11, 5) # Reduce the range of neighbors
55 param_grid = {'n_components': n_components, 'n_neighbors': n_neighbors}
```

```

57 # Custom scoring function for mean accuracy
58 def custom_score(estimator, X, y):
59     y_pred = estimator.predict(X)
60     return accuracy_score(y, y_pred)
61
62 random_search = RandomizedSearchCV(Isomap(), param_distributions=param_grid, n_iter=10, cv=3,
63                                     random_state=42, scoring=custom_score)
64 start_time = time.time()
65 random_search.fit(X_train, y_train)
66 end_time = time.time()
67 execution_time_with_isomap = end_time - start_time
68 best_params = random_search.best_params_
69 print("Best Parameters for ISOMAP:", best_params)
70 X_train_isomap = random_search.transform(X_train)
71 X_test_isomap = random_search.transform(X_test)
72 y_pred_with_isomap, _ = evaluate_knn(X_train_isomap, y_train, X_test_isomap, y_test)
73 print("ISOMAP Execution Time: {:.2f} seconds".format(execution_time_with_isomap))
74 print("\n")
75
75 # Confusion matrix without ISOMAP
76 plt.figure(figsize=(10, 8))
77 cm_no_isomap = confusion_matrix(y_test, y_pred_no_isomap)
78 cm_percent_no_isomap = cm_no_isomap / cm_no_isomap.sum(axis=1)[:, np.newaxis] * 100 # Calculate
79             percentages
80 sns.heatmap(cm_percent_no_isomap, annot=True, fmt=' .1f', cmap='Blues') # Display as floating-
81             point values with one decimal place
80 plt.title("Confusion Matrix without ISOMAP")
81 plt.savefig("confusion_matrix_no_isomap.pdf")
82 plt.show()
83
84 # Confusion matrix with ISOMAP
85 plt.figure(figsize=(10, 8))
86 cm_with_isomap = confusion_matrix(y_test, y_pred_with_isomap)
87 cm_percent_with_isomap = cm_with_isomap / cm_with_isomap.sum(axis=1)[:, np.newaxis] * 100 # Calculate
88             percentages
88 sns.heatmap(cm_percent_with_isomap, annot=True, fmt=' .1f', cmap='Blues') # Display as floating-
89             point values with one decimal place
89 plt.title("Confusion Matrix with ISOMAP")
90 plt.savefig("confusion_matrix_with_isomap.pdf")
91 plt.show()
92
93 # Clear unnecessary data from memory
94 del X, y, subset_indices, X_train_isomap, X_test_isomap, random_search

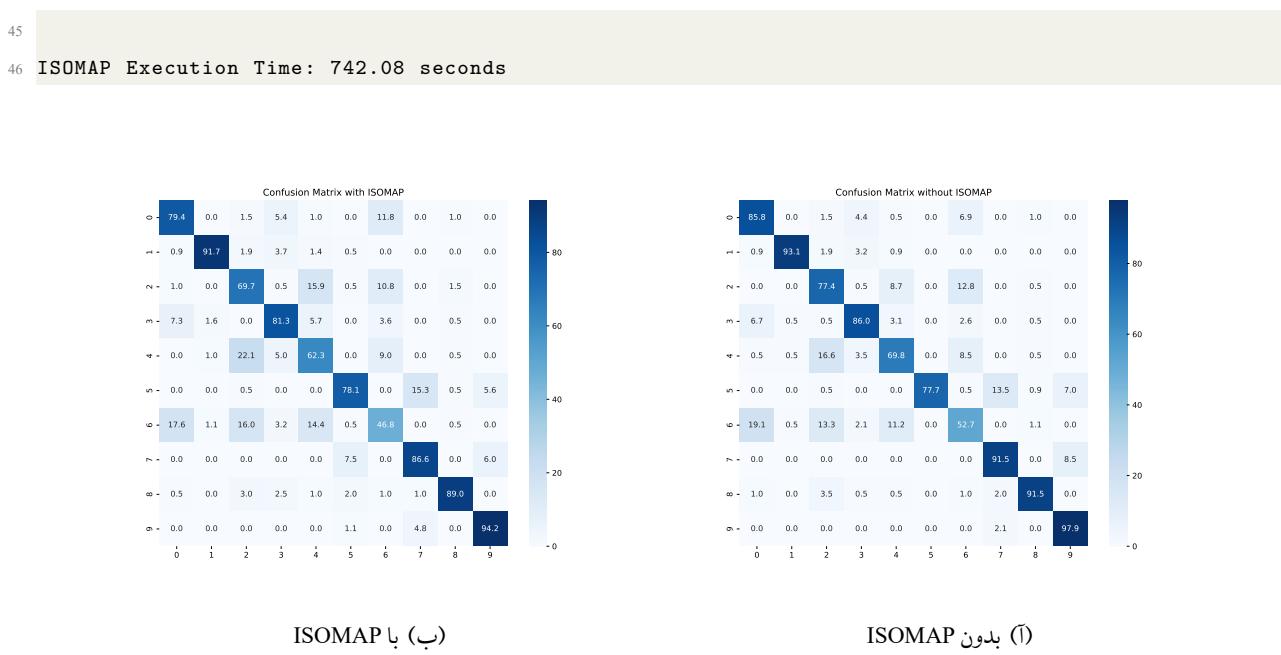
```

Program 5: ISOMAP + kNN Results

```

1 Without ISOMAP:
2 KNN Classification Report:
3
4
5             precision    recall   f1-score   support
6
7                 0       0.76      0.86      0.81      204
8                 1       0.99      0.93      0.96      216
9                 2       0.67      0.77      0.72      195
10                3       0.85      0.86      0.86      193
11                4       0.74      0.70      0.72      199
12                5       1.00      0.78      0.87      215
13                6       0.61      0.53      0.56      188
14                7       0.83      0.92      0.87      201
15                8       0.95      0.92      0.93      200
16                9       0.85      0.98      0.91      189
17
18             accuracy          0.82      2000
19             macro avg          0.83      0.82      2000
20 weighted avg          0.83      0.82      0.82      2000
21
22
23 With ISOMAP:
24 Best Parameters for ISOMAP: {'n_neighbors': 5, 'n_components': 10}
25 KNN Classification Report:
26
27
28             precision    recall   f1-score   support
29
30                 0       0.76      0.79      0.78      204
31                 1       0.97      0.92      0.94      216
32                 2       0.61      0.70      0.65      195
33                 3       0.79      0.81      0.80      193
34                 4       0.62      0.62      0.62      199
35                 5       0.88      0.78      0.83      215
36                 6       0.55      0.47      0.51      188
37                 7       0.80      0.87      0.83      201
38                 8       0.95      0.89      0.92      200
39                 9       0.88      0.94      0.91      189
40
41             accuracy          0.78      2000
42             macro avg          0.78      0.78      2000
43 weighted avg          0.78      0.78      0.78      2000
44
45 KNN Execution Time: 0.82 seconds

```



شکل ۶: نتایج ماتریس درهم‌ریختگی استفاده و عدم استفاده از ISOMAP.

۳.۱ پاسخ قسمت ۳ - رمزگذار-رمزگشا

ابتدا دستوراتی می‌نویسیم تا یک رمزگذار-رمزگشا با لایه‌های تمام‌امتصل را پیاده‌سازی کنیم. این دستورات یک مدل رمزگذار-رمزگشا (DAE) را تعریف و آموزش می‌دهند و سپس از طریق استفاده از مدل رمزگشا، داده‌ها را به فضای بعد کاهش یافته تبدیل کرده و با استفاده از الگوریتم kNN (نزدیکترین همسایه)، مدل‌های مختلفی با ابعاد بعدی کاهش یافته را آموزش می‌دهد. در نهایت، عملکرد هر مدل کاهش بعدی را ارزیابی می‌کند. در این دستورات ابتدا کتابخانه‌های ضروری فراخوانی می‌شوند. سپس بررسی می‌شود که آیا دستگاه GPU موجود است یا خیر و به متناسب با آن پیامی چاپ می‌شود. در ادامه، یک کلاس با نام DAE تعریف می‌شود که یک مدل رمزگذار-رمزگشا است. این کلاس شامل توابع مختلف برای تعریف مدل و آموزش آن است. لایه‌ها تمام‌امتصل (چگال یا Dense) در بخش تعریف شبکه با تابع فعال‌سازی `relu` و تابع مقداردهی `glorot_uniform` به اندازه نصف ابعاد ورودی ساخته می‌شوند و با لایه نرم‌مال‌سازی `Model` دسته برای نرم‌مال‌سازی خروجی قبلی و لایه فعال‌سازی `relu` ترکیب می‌شوند. تابع فعال‌ساز آخرین لایه تمام‌امتصل `sigmoid` است. در نهایت، مدل اتوانکو در با استفاده از `Model` از ورودی `input_img` به خروجی `decoded` تعریف می‌شود و بهینه‌ساز `Adam` و تابع هزینه `MSE` استفاده می‌شود. همچنین، یک مدل انکودر نیز با استفاده از `Model` از ورودی `input_img` به خروجی `encoded` تعریف می‌شود. تابع `fit` برای آموزش مدل رمزگذار-رمزگشا استفاده می‌شود. ابتدا داده‌های آموزش را به دو بخش آموزش و اعتبارسنجی تقسیم می‌کند. سپس با استفاده از الگوریتم Adam، مدل را آموزش می‌دهد. نمودار خطای آموزش و اعتبارسنجی را رسم می‌کند و در نهایت مدل آموزش دیده را بر می‌گرداند. برخی متدها مانند توقف زودهنگام در این قسمت اضافه شده‌اند و اطلاعات آموزش هم به صورت فایل `csv` ذخیره می‌شوند. در انتهای، نمودار منحنی هزینه رسم می‌شود و مدل به عنوان خروجی برگردانده می‌شود. تابع `transform` برای تبدیل داده‌ها به فضای بعد کاهش یافته با استفاده از مدل رمزگشا استفاده می‌شود. داده‌های ورودی را تبدیل می‌کند و نتیجه را بر می‌گرداند. با تعریف این کلاس و دستورات، مجموعه داده Fashion MNIST را بارگیری کرده و آن را به دو بخش آموزش و

آزمون تقسیم می‌کنیم. سپس مطابق توضیحات قبلی داده‌ها را پیش‌بردازش می‌کنیم. در ادامه، در این قسمت، یک لیست از اندازه‌های مخفی مختلف (latent_dim_values) تعریف می‌شود و برای هر اندازه مخفی، یک نمونه از کلاس DAE ساخته و آموزش داده می‌شود. همچنین برای هر مدل DAE، داده‌ها به فضای بُعد کاهش یافته تبدیل می‌شوند و یک مدل kNN روی فضای بُعد کاهش یافته آموزش داده می‌شود و عملکرد آن روی داده‌های تست ارزیابی می‌شود و نتایج مختلف گزارش می‌شود. دستورات مربوطه در برنامه ۶ آورده شده و نتایج در برنامه ۷، شکل ۷ و شکل ۸ نشان داده شده است.

Program 6: Dense AutoEncoder + kNN Implementation

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.model_selection import train_test_split
4 from sklearn.neighbors import KNeighborsClassifier
5 from sklearn.metrics import confusion_matrix
6 import seaborn as sns
7 from tensorflow.keras.datasets import fashion_mnist
8 from tensorflow.keras.models import Model
9 from tensorflow.keras.layers import Input, Dense, BatchNormalization, Activation
10 from tensorflow.keras.optimizers import Adam
11 from tensorflow.keras import callbacks
12 import tensorflow as tf
13
14 # Check if GPU is available
15 if tf.test.gpu_device_name() == '':
16     print('GPU device not found. Using CPU.')
17 else:
18     print('GPU device found. Using GPU.')
19
20 class DAE:
21     def __init__(self, input_dim, batch_size, latent_dim):
22         self.input_dim = input_dim
23         self.batch_size = batch_size
24         self.latent_dim = latent_dim
25         input_img = Input(shape=(self.input_dim, ))
26
27         # 'encoded' is the encoded representation of the input
28         encoded = Dense(int(self.input_dim / 2), kernel_initializer='glorot_uniform')(input_img)
29         encoded = BatchNormalization()(encoded)
30         encoded = Activation('relu')(encoded)
31
32         encoded = Dense(int(self.input_dim / 4), kernel_initializer='glorot_uniform')(encoded)
33         encoded = BatchNormalization()(encoded)
34         encoded = Activation('relu')(encoded)
35
36         encoded = Dense(int(self.input_dim / 8), kernel_initializer='glorot_uniform')(encoded)
37         encoded = BatchNormalization()(encoded)
38         encoded = Activation('relu')(encoded)

```

```

39
40     encoded = Dense(self.latent_dim, activation='linear')(encoded)
41
42     # 'decoded' is the lossy reconstruction of the input
43     decoded = Dense(int(self.input_dim / 8), kernel_initializer='glorot_uniform')(encoded)
44     decoded = BatchNormalization()(decoded)
45     decoded = Activation('relu')(decoded)
46
47     decoded = Dense(int(self.input_dim / 4), kernel_initializer='glorot_uniform')(decoded)
48     decoded = BatchNormalization()(decoded)
49     decoded = Activation('relu')(decoded)
50
51     decoded = Dense(int(self.input_dim / 2), kernel_initializer='glorot_uniform')(decoded)
52     decoded = BatchNormalization()(decoded)
53     decoded = Activation('relu')(decoded)
54
55     decoded = Dense(self.input_dim, activation='sigmoid', kernel_initializer='glorot_uniform')(decoded)
56
57     self.autoencoder = Model(inputs=input_img, outputs=decoded)
58     self.autoencoder.compile(optimizer=Adam(), loss='mse')
59     self.encoder = Model(inputs=input_img, outputs=encoded)
60
61 def fit(self, x_train):
62     x_train, x_valid = train_test_split(
63         x_train,
64         test_size=int(0.1 * x_train.shape[0] // self.batch_size * self.batch_size),
65         train_size=int(0.9 * x_train.shape[0] // self.batch_size * self.batch_size),
66         stratify=y_train)
67
68     with tf.device('/device:GPU:0'):
69         history = self.autoencoder.fit(
70             x_train,
71             x_train,
72             epochs=15,
73             batch_size=self.batch_size,
74             validation_data=(x_valid, x_valid),
75             verbose=1,
76             callbacks=[
77                 callbacks.EarlyStopping(
78                     monitor='val_loss',
79                     min_delta=0.0001,
80                     patience=10,
81                     restore_best_weights=True),
82                 callbacks.CSVLogger('training_log.csv')]

```

```

83         ])
84
85     # Plot and save loss curve
86     plt.figure(figsize=(8, 6))
87     plt.plot(history.history['loss'], label='Training Loss')
88     plt.plot(history.history['val_loss'], label='Validation Loss')
89     plt.xlabel('Epoch')
90     plt.ylabel('Loss')
91     plt.title('Loss Curve')
92     plt.legend()
93     plt.savefig('loss_curve.pdf')
94     plt.show()
95     return self
96
97 def transform(self, x):
98     with tf.device('/device:GPU:0'):
99         prediction = self.encoder.predict(x)
100    return prediction.reshape((len(prediction), np.prod(prediction.shape[1:])))
101
102 # Load the fashion_mnist dataset
103 (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
104
105 # Preprocess the data
106 x_train = x_train.reshape(-1, 784) / 255.0
107 x_test = x_test.reshape(-1, 784) / 255.0
108
109 # Initialize and train the autoencoder
110 latent_dim_values = [32, 64] # Different values for latent dimension
111 classifiers = []
112 for latent_dim in latent_dim_values:
113     dae = DAE(input_dim=784, batch_size=64, latent_dim=latent_dim)
114     dae.fit(x_train)
115     x_train_encoded = dae.transform(x_train)
116     x_test_encoded = dae.transform(x_test)
117
118     # Train a KNN classifier on the reduced dimensional space
119     knn = KNeighborsClassifier(n_neighbors=10)
120     knn.fit(x_train_encoded, y_train)
121     classifiers.append((knn, latent_dim))
122
123 # Evaluate and plot confusion matrix for each classifier
124 for knn, latent_dim in classifiers:
125     x_test_encoded = dae.transform(x_test)
126     x_test_encoded = x_test_encoded[:, :latent_dim] # Adjust the number of features
127     y_pred = knn.predict(x_test_encoded)

```

```

128     cm = confusion_matrix(y_test, y_pred)

129

130     # Plot confusion matrix as heatmap
131     plt.figure(figsize=(8, 6))
132     sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
133     plt.xlabel('Predicted label')
134     plt.ylabel('True label')
135     plt.title(f'Confusion Matrix (Latent Dim: {latent_dim})')
136     plt.savefig(f'confusion_matrix_latent_dim_{latent_dim}.pdf')
137     plt.show()

138

139 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
140

141 # Evaluate and print performance metrics for each classifier
142 for knn, latent_dim in classifiers:
143     x_test_encoded = dae.transform(x_test)
144     x_test_encoded = x_test_encoded[:, :latent_dim] # Adjust the number of features
145     y_pred = knn.predict(x_test_encoded)

146

147     accuracy = accuracy_score(y_test, y_pred)
148     precision = precision_score(y_test, y_pred, average='weighted')
149     recall = recall_score(y_test, y_pred, average='weighted')
150     f1 = f1_score(y_test, y_pred, average='weighted')

151

152     print(f"Performance metrics for Latent Dim: {latent_dim}")
153     print(f"Accuracy: {accuracy:.4f}")
154     print(f"Precision: {precision:.4f}")
155     print(f"Recall: {recall:.4f}")
156     print(f"F1 Score: {f1:.4f}")
157     print("-----")

```

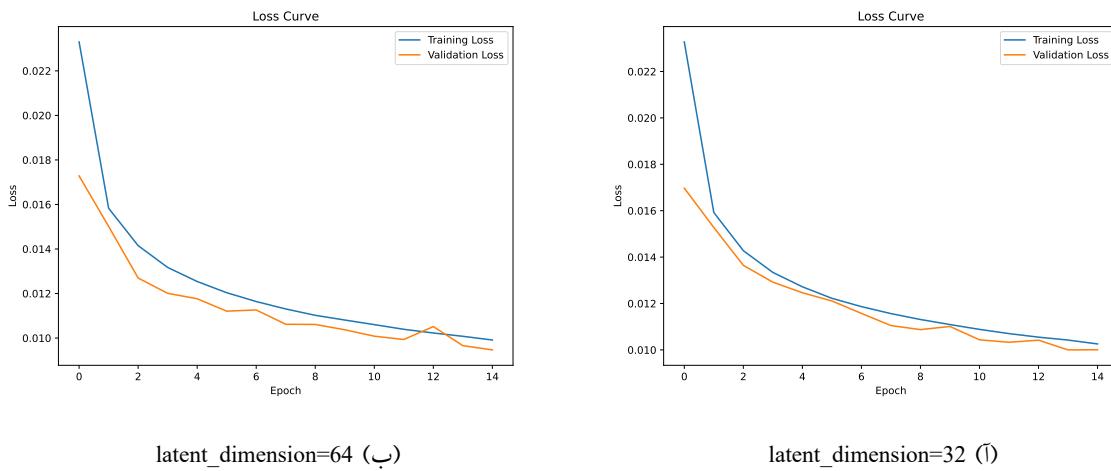
Program 7: Dense AutoEncoder + kNN Results

```

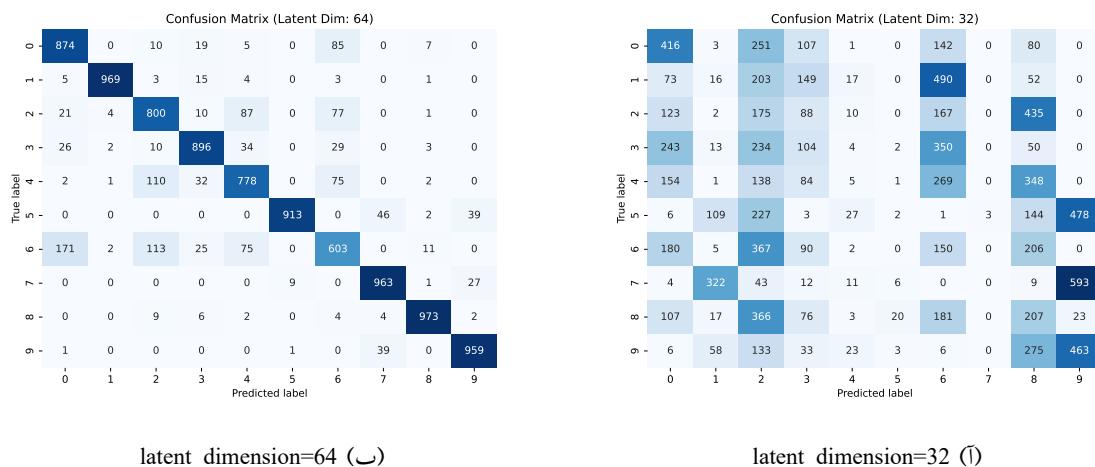
1 Performance metrics for Latent Dim: 32
2 Accuracy: 0.1538
3 Precision: 0.1172
4 Recall: 0.1538
5 F1 Score: 0.1243
6 -----
7 Performance metrics for Latent Dim: 64
8 Accuracy: 0.8728
9 Precision: 0.8726
10 Recall: 0.8728
11 F1 Score: 0.8719

```

در ادامه دستوراتی می‌نویسیم تا یک رمزگذار-رمزگشا با لایه‌های کانولوشنی را پیاده‌سازی کنیم. این دستورات یک مدل رمزگذار-رمزگشا (CAE) را تعریف و آموزش می‌دهند و سپس از طریق استفاده از مدل رمزگشا، داده‌ها را به فضای بعد کاهش



شکل ۷: نمودار تابع اتلاف آموزش مدل رمزگذار-رمزگشای تمام‌امتصل.



شکل ۸: نتیجه طبقه‌بندی در حالت رمزگذار-رمزگشای تمام‌امتصل.

یافته تبدیل کرده و با استفاده از الگوریتم kNN (نژدیک‌ترین همسایه)، مدل‌های مختلفی با ابعاد بعدی کاهش یافته را آموزش می‌دهد. در نهایت، عملکرد هر مدل کاهش بعدی را ارزیابی می‌کند. در این دستورات ابتدا کتابخانه‌های ضروری فراخوانی می‌شوند. سپس بررسی می‌شود که آیا دستگاه GPU موجود است یا خیر و به متناسب با آن پیامی چاپ می‌شود. در ادامه، یک کلاس با نام CAE تعریف می‌شود که یک مدل رمزگذار-رمزگشا است. این کلاس شامل توابع مختلف برای تعریف مدل و آموزش آن است. لایه‌ها کانولوشنی در بخش تعریف شبکه روی هم پشته شده‌اند و از روش‌ها و لایه‌هایی مانند اجماع بیشینه هم استفاده شده است. در نهایت، مدل اتوانکودر با استفاده از Model از ورودی input_img به خروجی decoded تعريف می‌شود و بهینه‌ساز Adam و تابع هزینه MSE استفاده می‌شود. همچنین، یک مدل انکودر نیز با استفاده از Model از ورودی encoded تعريف می‌شود. تابع fit برای آموزش مدل رمزگذار-رمزگشا استفاده می‌شود. ابتدا داده‌های آموزش را به دو بخش آموزش و اعتبارسنجی تقسیم می‌کند. سپس با استفاده از الگوریتم Adam، مدل را آموزش می‌دهد. نمودار

خطای آموزش و اعتبارسنجی را رسم می‌کند و در نهایت مدل آموزش دیده را بر می‌گرداند. برخی متدها مانند توقف زودهنگام در این قسمت اضافه شده‌اند و اطلاعات آموزش هم به صورت فایل csv ذخیره می‌شوند. در انتهای، نمودار منحنی هزینه رسم می‌شود و مدل به عنوان خروجی برگردانده می‌شود.تابع transform برای تبدیل داده‌ها به فضای بعد کاهش یافته با استفاده از مدل رمزگشا استفاده می‌شود. داده‌های ورودی را تبدیل می‌کند و نتیجه را بر می‌گرداند. با تعریف این کلاس و دستورات، مجموعه داده Fashion MNIST را بارگیری کرده و آن را به دو بخش آموزش و آزمون تقسیم می‌کنیم. سپس مطابق توضیحات قبلی داده‌ها را پیش‌پردازش می‌کنیم. در ادامه، در این قسمت، یک لیست از اندازه‌های مخفی مختلف (latent_dim_values) تعريف می‌شود و برای هر اندازه مخفی، یک نمونه از کلاس DAE ساخته و آموزش داده می‌شود. همچنین برای هر مدل DAE داده‌ها به فضای بعد کاهش یافته تبدیل می‌شوند و یک مدل kNN روی فضای بعد کاهش یافته آموزش داده می‌شود و عملکرد آن روی داده‌های تست ارزیابی می‌شود و نتایج مختلف گزارش می‌شود. دستورات مربوطه در برنامه ۸ آورده شده و نتایج در برنامه ۹، شکل ۱۰ نشان داده شده است. مشاهده می‌شود که نتایج در حالت استفاده از تعداد مخفی ۶۴ در هر دو حالت چگال و کانولوشنی بهتر بوده و در مجموع نتایج استفاده از لایه‌های تماماً متصل (چگال) کمی بهتر است.

Program 8: Conv AutoEncoder + kNN Implementation

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.model_selection import train_test_split
4 from sklearn.neighbors import KNeighborsClassifier
5 from sklearn.metrics import confusion_matrix
6 import seaborn as sns
7 from tensorflow.keras.datasets import fashion_mnist
8 from tensorflow.keras.models import Model
9 from tensorflow.keras.layers import Input, Dense, BatchNormalization, Activation, Conv2D,
   MaxPooling2D, Flatten, Reshape
10 from tensorflow.keras.optimizers import Adam
11 from tensorflow.keras import callbacks
12 import tensorflow as tf
13
14 # Check if GPU is available
15 if tf.test.gpu_device_name() == '':
16     print('GPU device not found. Using CPU.')
17 else:
18     print('GPU device found. Using GPU.')
19
20 class CE:
21     def __init__(self, input_shape, batch_size, latent_dim):
22         self.input_shape = input_shape
23         self.batch_size = batch_size
24         self.latent_dim = latent_dim
25         input_img = Input(shape=self.input_shape)
26
27         # Encoder
28         encoded = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(
            input_img)

```

```

29     encoded = MaxPooling2D(pool_size=(2, 2), padding='same')(encoded)
30     encoded = Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding='same')(
31         encoded)
32     encoded = MaxPooling2D(pool_size=(2, 2), padding='same')(encoded)
33     encoded = Flatten()(encoded)
34     encoded = Dense(self.latent_dim, activation='linear')(encoded)
35
36     # Decoder
37     decoded = Dense(7 * 7 * 8, activation='relu')(encoded)
38     decoded = Reshape((7, 7, 8))(decoded)
39     decoded = Conv2D(filters=8, kernel_size=(3, 3), activation='relu', padding='same')(
40         decoded)
41     decoded = UpSampling2D((2, 2))(decoded)
42     decoded = Conv2D(filters=16, kernel_size=(3, 3), activation='relu', padding='same')(
43         decoded)
44     decoded = UpSampling2D((2, 2))(decoded)
45     decoded = Conv2D(filters=1, kernel_size=(3, 3), activation='sigmoid', padding='same')(
46         decoded)
47
48     self.autoencoder = Model(inputs=input_img, outputs=decoded)
49     self.autoencoder.compile(optimizer=Adam(), loss='mse')
50     self.encoder = Model(inputs=input_img, outputs=encoded)
51
52
53     def fit(self, x_train):
54         x_train, x_valid = train_test_split(
55             x_train,
56             test_size=int(0.1 * x_train.shape[0] // self.batch_size * self.batch_size),
57             train_size=int(0.9 * x_train.shape[0] // self.batch_size * self.batch_size),
58             stratify=y_train)
59
60         with tf.device('/device:GPU:0'):
61             history = self.autoencoder.fit(
62                 x_train,
63                 x_train,
64                 epochs=15,
65                 batch_size=self.batch_size,
66                 validation_data=(x_valid, x_valid),
67                 verbose=1,
68                 callbacks=[
69                     callbacks.EarlyStopping(
70                         monitor='val_loss',
71                         min_delta=0.0001,
72                         patience=10,
73                         restore_best_weights=True),
74                     callbacks.CSVLogger('training_log.csv')]
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
927
928
929
929
930
931
932
933
934
935
936
937
937
938
939
939
940
941
942
943
944
945
945
946
947
947
948
949
949
950
951
952
953
954
955
956
956
957
958
958
959
959
960
961
962
963
964
964
965
965
966
966
967
967
968
968
969
969
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
166
```

```

70         ])
71
72     # Plot and save loss curve
73     plt.figure(figsize=(8, 6))
74     plt.plot(history.history['loss'], label='Training Loss')
75     plt.plot(history.history['val_loss'], label='Validation Loss')
76     plt.xlabel('Epoch')
77     plt.ylabel('Loss')
78     plt.title('Loss Curve')
79     plt.legend()
80     plt.savefig('loss_curve.pdf')
81     plt.show()
82     return self
83
84 def transform(self, x):
85     with tf.device('/device:GPU:0'):
86         prediction = self.encoder.predict(x)
87     return prediction
88
89 # Load the fashion_mnist dataset
90 (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
91
92 # Preprocess the data
93 x_train = x_train.reshape(-1, 28, 28, 1) / 255.0
94 x_test = x_test.reshape(-1, 28, 28, 1) / 255.0
95
96 # Initialize and train the autoencoder
97 latent_dim_values = [32, 64] # Different values for latent dimension
98 classifiers = []
99 for latent_dim in latent_dim_values:
100     ce = CE(input_shape=(28, 28, 1), batch_size=64, latent_dim=latent_dim)
101     ce.fit(x_train)
102     x_train_encoded = ce.transform(x_train)
103     x_test_encoded = ce.transform(x_test)
104
105     # Train a KNN classifier on the reduced dimensional space
106     knn = KNeighborsClassifier(n_neighbors=10)
107     knn.fit(x_train_encoded, y_train)
108     classifiers.append((knn, latent_dim))
109
110 # Evaluate and plot confusion matrix for each classifier
111 for knn, latent_dim in classifiers:
112     x_test_encoded = ce.transform(x_test)
113     x_test_encoded = x_test_encoded[:, :latent_dim] # Adjust the number of features
114     y_pred = knn.predict(x_test_encoded)

```

```

115     cm = confusion_matrix(y_test, y_pred)

116

117     # Plot confusion matrix as heatmap
118     plt.figure(figsize=(8, 6))
119     sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
120     plt.xlabel('Predicted label')
121     plt.ylabel('True label')
122     plt.title(f'Confusion Matrix (Latent Dim: {latent_dim})')
123     plt.savefig(f'confusion_matrix_latent_dim_{latent_dim}.pdf')
124     plt.show()

125

126 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
127

128 # Evaluate and print performance metrics for each classifier
129 for knn, latent_dim in classifiers:
130     x_test_encoded = ce.transform(x_test)
131     x_test_encoded = x_test_encoded[:, :latent_dim] # Adjust the number of features
132     y_pred = knn.predict(x_test_encoded)

133

134     accuracy = accuracy_score(y_test, y_pred)
135     precision = precision_score(y_test, y_pred, average='weighted')
136     recall = recall_score(y_test, y_pred, average='weighted')
137     f1 = f1_score(y_test, y_pred, average='weighted')

138

139     print(f"Performance metrics for Latent Dim: {latent_dim}")
140     print(f"Accuracy: {accuracy:.4f}")
141     print(f"Precision: {precision:.4f}")
142     print(f"Recall: {recall:.4f}")
143     print(f"F1 Score: {f1:.4f}")
144     print("-----")

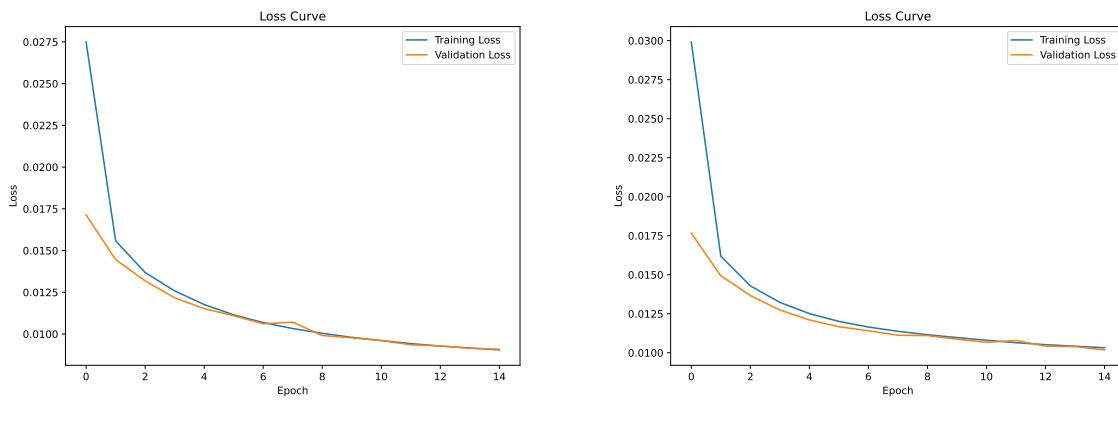
```

Program 9: Conv AutoEncoder + kNN Results

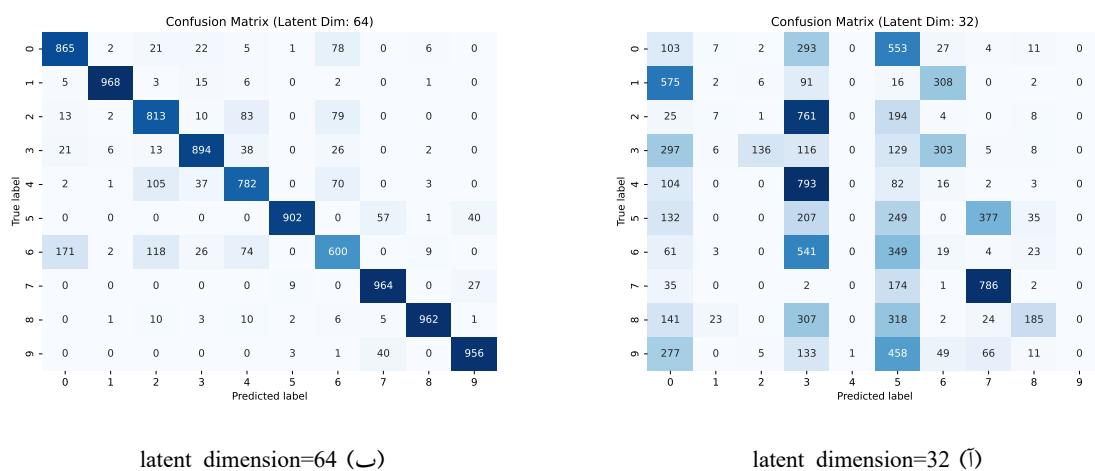
```

1 Performance metrics for Latent Dim: 32
2 Accuracy: 0.1461
3 Precision: 0.1530
4 Recall: 0.1461
5 F1 Score: 0.1279
6 -----
7 Performance metrics for Latent Dim: 64
8 Accuracy: 0.8706
9 Precision: 0.8706
10 Recall: 0.8706
11 F1 Score: 0.8697

```



شکل ۹: نمودار تابع اتلاف آموزش مدل رمزگذار-رمزگشای کانولوشنی.

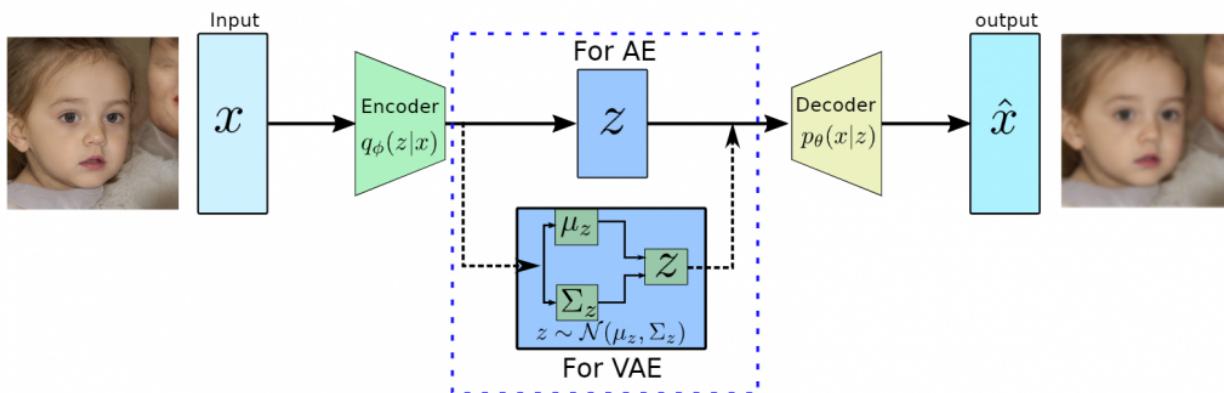


شکل ۱۰: نتیجه طبقه‌بندی در حالت رمزگذار-رمزگشای کانولوشنی.

۴.۱ پاسخ قسمت ۴ - خودرمنگذار متغیر (Variational AutoEncoder)

تفاوت بین خودرمزگذار متغیر با نوع عادی آن، به فضای نهفته متفاوت آن بازمی‌گردد. در این خودرمزگذارها، فضای نهفته به صورت یک توزیع با میانگین و انحراف یادگیری شده، به جای فقط مجموعه‌ای از اعداد، نمایش داده می‌شود. این توزیع به صورت معمول گاووسی چندمتغیره در نظر گرفته می‌شود. عملکرد VAE بر مبنای دو بخش اصلی است: بخشی که به آن Latent (آنکودر گفته می‌شود و بخشی که به آن دیکودر گفته می‌شود. این دو بخش با همکاری یکدیگر، یک فضای نهان (Space) را تولید می‌کنند که داده‌ها را در آن فضای نمایش می‌دهند. در واقع در این مرحله، یک نمایش از داده‌ها در فضای نهان بدست می‌آید که به شکل یک توزیع احتمالی چگالی احتمال نشان داده می‌شود. این توزیع چگالی احتمال در واقع احتمال وجود هر نمونه در فضای نهان را بیان می‌کند. در بخش انکودر، ورودی (مانند تصویر) به یک فضای نهان با ابعاد کمتر تبدیل می‌شود. این فضای نهان معمولاً یک فضای توزیع احتمالاتی است که مشخص می‌کند که هر نقطه در این فضای نمایش دهنده

چه نوع توزیع در فضای داده‌هاست. این توزیع معمولاً یک توزیع گاووسی چندمتغیره است. به جای انتخاب نقاط دقیق از توزیع چگالی احتمال در فضای نهان (که کار بسیار پیچیده‌تری است)، از تکنیک نمونه‌برداری از این توزیع استفاده می‌شود تا نمونه‌های مختلفی از فضای نهان بدست آید. مهترین ویژگی VAE در بخش انکودر، اعمال محدودیت‌هایی روی فضای نهان است. این محدودیت‌ها توسط یک توزیع گاووسی با میانگین صفر و واریانس یک اعمال می‌شوند. به این ترتیب، نقاطی که در فضای نهان به دست می‌آیند، از توزیع گاووسی با میانگین صفر و واریانس یک نمونه‌گیری می‌شوند. در بخش دیکودر، نقاطی که در فضای نهان تولید شده‌اند به داده‌های قابل خواندن تبدیل می‌شوند. برای این کار، نقاط فضای نهان با استفاده از تابعی به نام دیکودر به داده‌های خروجی مطابقت می‌یابند. آموزش یک مدل VAE از طریق بهینه‌سازی ELBO صورت می‌گیرد. ELBO مقدار لگاریتم احتمال داده‌های ورودی به شرط نمونه‌گیری از توزیع فضای نهان است. با داشتن ELBO، می‌توانیم مدل را آموزش دهیم تا فضای نهان را به نحوی یاد بگیرد که توزیع داده‌های واقعی را به خوبی تقریب بزند. به عبارت دیگر، با استفاده از VAE می‌توانیم نقاط جدیدی در فضای نهان ایجاد کنیم و با تبدیل این نقاط به فضای داده‌ها، داده‌های جدیدی را تولید کنیم که به شکلی شبیه به داده‌های ورودی باشند. این ویژگی از VAE برای تولید تصاویر جدید، ایجاد واقعیت مجازی، توسعه بازی‌های ویدئویی و بسیاری از کاربردهای دیگر استفاده می‌شود. برای پیاده‌سازی اهداف مورد خواست سوال مشابه قسمت‌های



شکل ۱۱: خودرمزگذار متغیر (Variational AutoEncoder).

قبل ابتدا کتابخانه‌های ضروری را فراخوانی می‌کنیم. سپس مجموعه داده Fashion MNIST را بارگیری کرده و آن را به صورتی مناسب پیش‌پردازش می‌کنیم. در ادامه کلاس DVAE را تعریف می‌کنیم که یک مدل (VAE) است که فضای داده‌ها را با استفاده از فشرده‌سازی و تولید داده‌ها پیاده‌سازی می‌شود. این کلاس دارای متدهای مختلفی است. ابتدا متدهای `init` و `reconstruct` را تعریف می‌کنیم. ورودی‌های این متدها `input_dim` است که ابعاد ورودی را مشخص می‌کند، `batch_size` که اندازه دسته‌ها در آموزش مدل را تعیین می‌کند و `latent_dim` که ابعاد فضای نهان را مشخص می‌کند. در این متدها ورودی‌های مدل تعریف می‌شوند و سپس بخش انکودر ساخته می‌شود. این بخش شامل لایه‌های تمام‌امتصل (چگال یا Dense) است که با استفاده از تابع فعال‌سازی ReLU، ابعاد ورودی را به صورت تدریجی کاهش می‌دهند و در نهایت دو لایه چگال با ابعاد ویژگی فضای نهان `z_mean` و `z_log_var` را ایجاد می‌کنند. سپس یک لایه Sampling تعریف می‌شود که با دریافت میانگین `z_mean` و واریانس `z_log_var`، نمونه‌های تصادفی از فضای نهان را تولید می‌کند. در نهایت بخش دیکودر تعریف می‌شود که شامل لایه‌های تمام‌امتصل (چگال یا Dense) است که با استفاده از تابع فعال‌سازی ReLU، ابعاد فضای نهان را به صورت تدریجی افزایش می‌دهند و در نهایت ابعاد خروجی را به اندازه ورودی اولیه برمی‌گردانند. سپس مدل‌های انکودر و دیکودر به صورت جداگانه تعریف می‌شوند و مدل VAE نیز با استفاده از این دو مدل ساخته می‌شود. تابع هزینه نیز با استفاده از دو بخش

VAE با استفاده از اینتابع هزینه و بهینه‌ساز Adam کامپایل می‌شود. KL Loss و Reconstruction Loss متدهای fit است. ورودی‌های این متدها از x_train که داده‌های آموزشی هستند و epochs که تعداد دوره‌ها را تعیین کند. در این متدها از داده‌های آموزش داده می‌شود. تاریخچه مقادیر هزینه در طول آموزش نگهداری می‌شود و سپس نمودار تغییرات هزینه آموزش و اعتبارسنجی رسم می‌شود. در نهایت تابع plot_loss فراخوانی می‌شود تا نمودار ذخیره شود و نمایش داده شود. در ادامه متدها transform تعریف می‌شود. ورودی این متدها عبارتست از x که داده‌های ورودی هستند. این متدها با استفاده از مدل انکوادر ورودی‌های داده‌ها را به فضای نهان نگاشت می‌دهد و نگاشتها را بر می‌گرداند. متدهای plot_loss نام دارد، که تاریخچه مقادیر هزینه در طول آموزش است را دریافت می‌کند. این متدهای نمودار تغییرات هزینه آموزش و اعتبارسنجی را رسم می‌کند و آن را نمایش می‌دهد. ورودی‌های متدهای plot_scatter عبارتند از x_train که داده‌های آموزشی هستند و labels که برچسب‌های مربوط به داده‌های آموزشی است. این متدها با استفاده از مدل انکوادر داده‌های آموزشی را به فضای نهان نگاشت می‌دهد و سپس نمودار Scatter از داده‌ها را در فضای نهان رسم می‌کند. در واقع این متدهای پراکنده‌گی داده‌های آموزش را در فضای کاهش یافته نشان می‌دهد. از داده‌های آموزش استفاده می‌کند تا نمایش کاهش یافته آنها را با استفاده از شبکه کدگذار بدست آورد. سپس داده‌های آموزش را بر اساس ابعاد کاهش یافته رسم می‌کند و بر اساس برچسب‌ها رنگ‌آمیزی می‌کند. در ادامه فضای کاهش یافته با استفاده از الگوریتم kNN دسته‌بندی می‌شود. یک نمونه از KNeighborsClassifier با پارامتر تعداد همسایگان (n_neighbors) برابر با ۱۰ ساخته شده و روی داده‌های فضای کاهش یافته آموزش داده می‌شود. سپس پیش‌بینی‌های دسته‌بندی روی داده‌های فضای کاهش یافته انجام شده و در متغير knn_pred ذخیره می‌شود. سپس نتایج این طبقه‌بندی نمایش داده می‌شود. لازم به ذکر است که انکوادر و دیکوادر مدل VAE ذخیره هم می‌شوند. همچین با استفاده از دیکوادر مدل VAE، تصاویر جدیدی تولید می‌شود. ابتدا یک نمونه از نویز نرمال با ابعاد مناسب تولید می‌شود و سپس با استفاده از دیکوادر به تصاویر جدید تبدیل می‌شود. نتایج در متغير generated_images ذخیره می‌شود و به شکل یک شبکه تصاویر به اندازه ۱۰ در ۱۰ نمایش داده می‌شود. با ارائه این توضیحات دستورات مربوطه در برنامه ۱۰ و نتایج در برنامه ۱۱، شکل ۱۲ و شکل ۱۳ آورده شده است.

Program 10: Dense VAE + kNN Implementation

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import tensorflow as tf
6 from tensorflow import keras
7 from tensorflow.keras import layers
8 from sklearn.model_selection import train_test_split
9 from sklearn.neighbors import KNeighborsClassifier
10 from sklearn.metrics import classification_report, confusion_matrix
11 from sklearn.preprocessing import MinMaxScaler
12 import warnings
13 warnings.filterwarnings('ignore')
14 warnings.filterwarnings("ignore", category=Warning)
15 warnings.filterwarnings("default", category=Warning)
16
17 # Load the fashion MNIST dataset
18 fashion_mnist = keras.datasets.fashion_mnist
19 (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

```

```

20
21 # Preprocess the data
22 scaler = MinMaxScaler()
23 train_images = train_images.reshape(train_images.shape[0], -1) / 255.0
24 test_images = test_images.reshape(test_images.shape[0], -1) / 255.0
25
26 # Define the VAE class
27 class VAE:
28     def __init__(self, input_dim, batch_size, latent_dim):
29         self.input_dim = input_dim
30         self.batch_size = batch_size
31         self.latent_dim = latent_dim
32
33     # Encoder
34     inputs = keras.Input(shape=(self.input_dim,))
35     x = layers.Dense(int(self.input_dim / 2), activation='relu')(inputs)
36     x = layers.Dense(int(self.input_dim / 4), activation='relu')(x)
37     x = layers.Dense(int(self.input_dim / 8), activation='relu')(x)
38     z_mean = layers.Dense(self.latent_dim, name="z_mean")(x)
39     z_log_var = layers.Dense(self.latent_dim, name="z_log_var")(x)
40
41     # Sampling layer
42     def sampling(args):
43         z_mean, z_log_var = args
44         epsilon = tf.random.normal(shape=(tf.shape(z_mean)[0], self.latent_dim))
45         return z_mean + tf.exp(0.5 * z_log_var) * epsilon
46
47     z = layers.Lambda(sampling, name="z")([z_mean, z_log_var])
48
49     # Decoder
50     decoder_inputs = layers.Input(shape=(self.latent_dim,))
51     x = layers.Dense(int(self.input_dim / 8), activation='relu')(decoder_inputs)
52     x = layers.Dense(int(self.input_dim / 4), activation='relu')(x)
53     x = layers.Dense(int(self.input_dim / 2), activation='relu')(x)
54     outputs = layers.Dense(self.input_dim, activation='sigmoid')(x)
55
56     # Models
57     self.encoder = keras.Model(inputs, z_mean)
58     self.decoder = keras.Model(decoder_inputs, outputs)
59     self.vae = keras.Model(inputs, self.decoder(self.encoder(inputs)))
60
61     # Loss function
62     reconstruction_loss = tf.reduce_mean(
63         keras.losses.binary_crossentropy(inputs, self.decoder(z)), axis=-1
64     )

```

```

65     kl_loss = -0.5 * tf.reduce_mean(1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var),
66                                     axis=-1)
67
68     vae_loss = tf.reduce_mean(reconstruction_loss + kl_loss)
69     self.vae.add_loss(vae_loss)
70     self.vae.compile(optimizer='adam')
71
72
73
74     def fit(self, x_train, epochs):
75         history = self.vae.fit(x_train, x_train, epochs=epochs, batch_size=self.batch_size,
76                                validation_split=0.1)
77         self.plot_loss(history)
78
79
80     def transform(self, x):
81         return self.encoder.predict(x)
82
83
84     def plot_loss(self, history):
85         plt.figure(figsize=(8, 6))
86         plt.plot(history.history['loss'], label='Train Loss')
87         plt.plot(history.history['val_loss'], label='Validation Loss')
88         plt.xlabel('Epochs')
89         plt.ylabel('Loss')
90         plt.title('Training and Validation Loss')
91         plt.legend()
92         plt.savefig(f'loss_plot_{self.latent_dim}.pdf')
93         plt.show()
94
95
96     def plot_scatter(self, x_train, labels):
97         latent_space = self.transform(x_train)
98         plt.figure(figsize=(8, 6))
99         sns.scatterplot(x=latent_space[:, 0], y=latent_space[:, 1], hue=labels, palette='tab10')
100        plt.xlabel('Latent Dimension 1')
101        plt.ylabel('Latent Dimension 2')
102        plt.title('Scatter Plot of Training Data in Latent Space')
103        plt.savefig(f'scatter_plot_{self.latent_dim}.pdf')
104        plt.show()
105
106
107 # Task 1: Train VAE and classify reduced dimension space with KNN
108 latent_dims = [32, 64]
109
110 for latent_dim in latent_dims:
111     # Create VAE instance
112     vae = VAE(input_dim=train_images.shape[1], batch_size=32, latent_dim=latent_dim)
113
114     # Train VAE
115     vae.fit(train_images, epochs=15)

```

```

108     # Reduce dimensionality of test data
109     reduced_test_images = vae.transform(test_images)
110
111     # Classify reduced dimension space with KNN
112     knn = KNeighborsClassifier(n_neighbors=10)
113     knn.fit(reduced_test_images, test_labels)
114     knn_pred = knn.predict(reduced_test_images)
115
116     # Print classification report
117     print(f"Latent Dimension: {latent_dim}")
118     print(classification_report(test_labels, knn_pred))
119
120     # Plot confusion matrix
121     cm = confusion_matrix(test_labels, knn_pred)
122     df_cm = pd.DataFrame(cm, index=np.arange(10), columns=np.arange(10))
123     plt.figure(figsize=(8, 6))
124     sns.heatmap(df_cm, annot=True, fmt="d", cmap="Blues")
125     plt.xlabel("Predicted label")
126     plt.ylabel("True label")
127     plt.title("Confusion Matrix")
128     plt.savefig(f"confusion_matrix_{latent_dim}.pdf")
129     plt.show()
130
131     # Save VAE encoder and decoder networks
132     vae.encoder.save(f"encoder_{latent_dim}.h5")
133     vae.decoder.save(f"decoder_{latent_dim}.h5")
134
135     # Generate images using the decoder
136     n = 10
137     latent_samples = np.random.normal(size=(n*n, latent_dim))
138     generated_images = vae.decoder.predict(latent_samples)
139     generated_images = generated_images.reshape(-1, 28, 28)
140     plt.figure(figsize=(10, 10))
141     for i in range(n):
142         for j in range(n):
143             plt.subplot(n, n, i * n + j + 1)
144             plt.imshow(generated_images[i * n + j], cmap="gray")
145             plt.axis("off")
146     plt.savefig(f"generated_images_{latent_dim}.pdf")
147     plt.show()
148
149     # Plot scatter plot of training data in latent space
150     vae.plot_scatter(train_images, train_labels)

```

Program 11: Dense VAE + kNN Results

```

1 Latent Dimension: 32
2           precision    recall   f1-score   support
3
4          0       0.63      0.79      0.70      1000
5          1       0.88      0.79      0.83      1000
6          2       0.50      0.61      0.55      1000
7          3       0.64      0.73      0.68      1000
8          4       0.57      0.57      0.57      1000
9          5       0.78      0.84      0.81      1000
10         6       0.47      0.25      0.33      1000
11         7       0.82      0.79      0.80      1000
12         8       0.81      0.78      0.79      1000
13         9       0.89      0.85      0.87      1000
14
15     accuracy                           0.70      10000
16   macro avg       0.70      0.70      0.69      10000
17 weighted avg    0.70      0.70      0.69      10000
18 -----
19 Latent Dimension: 64
20           precision    recall   f1-score   support
21
22          0       0.71      0.79      0.74      1000
23          1       0.97      0.82      0.89      1000
24          2       0.51      0.65      0.57      1000
25          3       0.67      0.82      0.74      1000
26          4       0.54      0.55      0.54      1000
27          5       0.81      0.86      0.83      1000
28          6       0.49      0.25      0.33      1000
29          7       0.83      0.80      0.81      1000
30          8       0.91      0.92      0.92      1000
31          9       0.88      0.87      0.87      1000
32
33     accuracy                           0.73      10000
34   macro avg       0.73      0.73      0.72      10000
35 weighted avg    0.73      0.73      0.72      10000

```

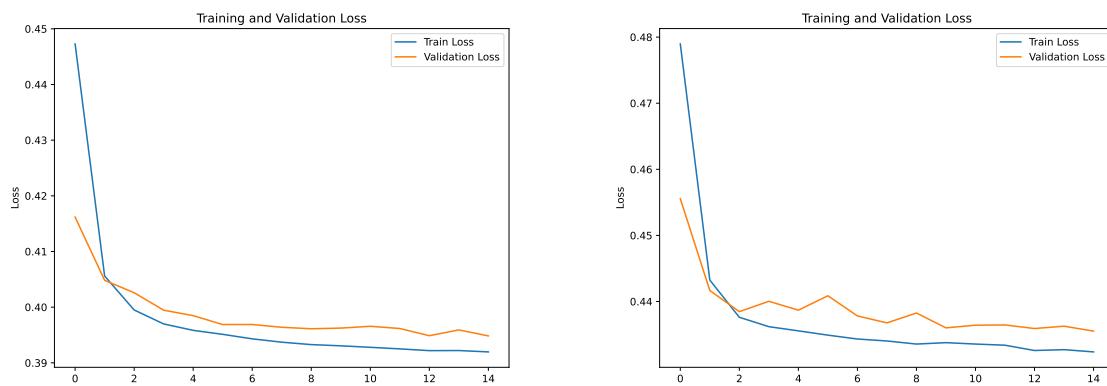
در ادامه کلاس VAE را با لایه‌های کانولشنی هم پیاده‌سازی می‌کنیم. توضیحات مطابق کلاس قبلی است و تنها لایه‌ها به صورت کانولشنی تغییر پیدا کرده‌اند. دستورات مربوطه در برنامه ۱۲ و نتایج در برنامه ۱۳، شکل ۱۴ و شکل ۱۵ آورده شده است.

Program 12: Conv VAE + kNN Implementation

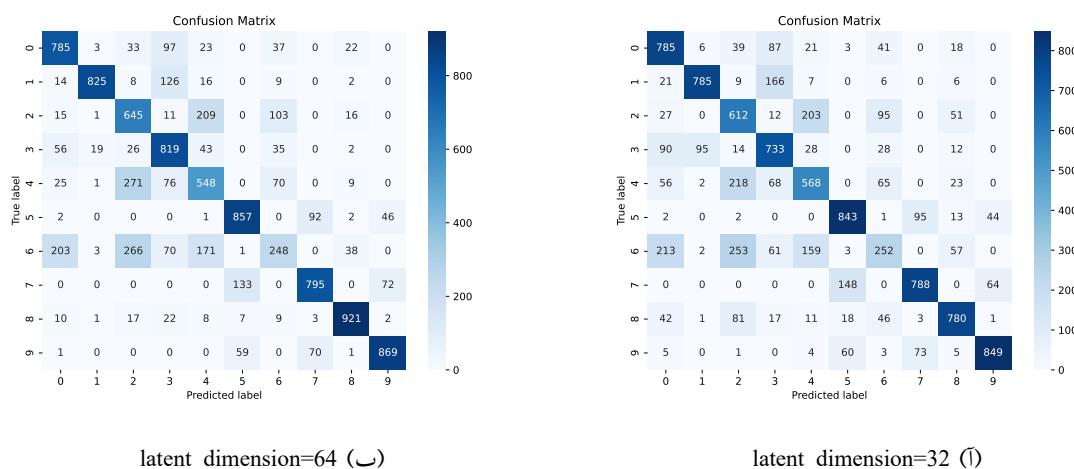
```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt

```



شکل ۱۲: نمودار تابع اتلاف آموزش مدل خودرمزگذار متغیر تماماً متصل.



شکل ۱۳: نتیجه طبقه‌بندی در حالت خودرمزگذار متغیر تماماً متصل.

```

4 import seaborn as sns
5 import tensorflow as tf
6 from tensorflow import keras
7 from tensorflow.keras import layers
8 from sklearn.model_selection import train_test_split
9 from sklearn.neighbors import KNeighborsClassifier
10 from sklearn.metrics import classification_report, confusion_matrix
11 from sklearn.preprocessing import MinMaxScaler
12 import warnings
13 warnings.filterwarnings('ignore')
14
15 # Load the fashion MNIST dataset

```

```

16 fashion_mnist = keras.datasets.fashion_mnist
17 (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
18
19 # Preprocess the data
20 scaler = MinMaxScaler()
21 train_images = train_images.reshape(train_images.shape[0], 28, 28, 1) / 255.0
22 test_images = test_images.reshape(test_images.shape[0], 28, 28, 1) / 255.0
23
24 # Define the CVAE class
25 class CVAE:
26     def __init__(self, input_shape, batch_size, latent_dim):
27         self.input_shape = input_shape
28         self.batch_size = batch_size
29         self.latent_dim = latent_dim
30
31     # Encoder
32     inputs = keras.Input(shape=self.input_shape)
33     x = layers.Conv2D(filters=32, kernel_size=3, strides=2, activation='relu', padding='same')(inputs)
34     x = layers.Conv2D(filters=64, kernel_size=3, strides=2, activation='relu', padding='same')(x)
35     x = layers.Flatten()(x)
36     x = layers.Dense(256, activation='relu')(x)
37     z_mean = layers.Dense(self.latent_dim, name="z_mean")(x)
38     z_log_var = layers.Dense(self.latent_dim, name="z_log_var")(x)
39
40     # Sampling layer
41     def sampling(args):
42         z_mean, z_log_var = args
43         epsilon = tf.random.normal(shape=(tf.shape(z_mean)[0], self.latent_dim))
44         return z_mean + tf.exp(0.5 * z_log_var) * epsilon
45
46     z = layers.Lambda(sampling, name="z")([z_mean, z_log_var])
47
48     # Decoder
49     decoder_inputs = layers.Input(shape=(self.latent_dim,))
50     x = layers.Dense(7 * 7 * 64, activation='relu')(decoder_inputs)
51     x = layers.Reshape((7, 7, 64))(x)
52     x = layers.Conv2DTranspose(filters=64, kernel_size=3, strides=2, activation='relu',
53                               padding='same')(x)
54     x = layers.Conv2DTranspose(filters=32, kernel_size=3, strides=2, activation='relu',
55                               padding='same')(x)
56     outputs = layers.Conv2DTranspose(filters=1, kernel_size=3, activation='sigmoid', padding=
57                                     'same')(x)
58
59

```

```

56     # Models
57
58     self.encoder = keras.Model(inputs, z_mean)
59     self.decoder = keras.Model(decoder_inputs, outputs)
60     self.vae = keras.Model(inputs, self.decoder(self.encoder(inputs)))
61
62     # Loss function
63     reconstruction_loss = tf.reduce_mean(
64         keras.losses.binary_crossentropy(inputs, self.decoder(z)), axis=[1, 2]
65     )
66
67     kl_loss = -0.5 * tf.reduce_mean(1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var),
68     axis=-1)
69
70     vae_loss = tf.reduce_mean(reconstruction_loss + kl_loss)
71     self.vae.add_loss(vae_loss)
72     self.vae.compile(optimizer='adam')
73
74 def fit(self, x_train, epochs):
75     history = self.vae.fit(x_train, x_train, epochs=epochs, batch_size=self.batch_size,
76     validation_split=0.1)
77     self.plot_loss(history)
78
79 def transform(self, x):
80     return self.encoder.predict(x)
81
82 def plot_loss(self, history):
83     plt.figure(figsize=(8, 6))
84     plt.plot(history.history['loss'], label='Train Loss')
85     plt.plot(history.history['val_loss'], label='Validation Loss')
86     plt.xlabel('Epochs')
87     plt.ylabel('Loss')
88     plt.title('Training and Validation Loss')
89     plt.legend()
90     plt.savefig(f'loss_plot_{self.latent_dim}.pdf')
91     plt.show()
92
93 def plot_scatter(self, x_train, labels):
94     latent_space = self.transform(x_train)
95     plt.figure(figsize=(8, 6))
96     sns.scatterplot(x=latent_space[:, 0], y=latent_space[:, 1], hue=labels, palette='tab10')
97     plt.xlabel('Latent Dimension 1')
98     plt.ylabel('Latent Dimension 2')
99     plt.title('Scatter Plot of Training Data in Latent Space')
100    plt.savefig(f'scatter_plot_{self.latent_dim}.pdf')
101    plt.show()
102
103 # Task 1: Train CVAE and classify reduced dimension space with KNN

```

```

99 latent_dims = [32, 64]
100
101 for latent_dim in latent_dims:
102     # Create CVAE instance
103     cvae = CVAE(input_shape=(28, 28, 1), batch_size=32, latent_dim=latent_dim)
104
105     # Train CVAE
106     cvae.fit(train_images, epochs=15)
107
108     # Reduce dimensionality of test data
109     reduced_test_images = cvae.transform(test_images)
110
111     # Classify reduced dimension space with KNN
112     knn = KNeighborsClassifier(n_neighbors=10)
113     knn.fit(reduced_test_images, test_labels)
114     knn_pred = knn.predict(reduced_test_images)
115
116     # Print classification report
117     print(f"Latent Dimension: {latent_dim}")
118     print(classification_report(test_labels, knn_pred))
119
120     # Plot confusion matrix
121     cm = confusion_matrix(test_labels, knn_pred)
122     df_cm = pd.DataFrame(cm, index=np.arange(10), columns=np.arange(10))
123     plt.figure(figsize=(8, 6))
124     sns.heatmap(df_cm, annot=True, fmt="d", cmap="Blues")
125     plt.xlabel("Predicted label")
126     plt.ylabel("True label")
127     plt.title("Confusion Matrix")
128     plt.savefig(f"confusion_matrix_{latent_dim}.pdf")
129     plt.show()
130
131     # Save CVAE encoder and decoder networks
132     cvae.encoder.save(f"encoder_{latent_dim}.h5")
133     cvae.decoder.save(f"decoder_{latent_dim}.h5")
134
135     # Generate images using the decoder
136     n = 10
137     latent_samples = np.random.normal(size=(n * n, latent_dim))
138     generated_images = cvae.decoder.predict(latent_samples)
139     generated_images = generated_images.reshape(-1, 28, 28)
140     plt.figure(figsize=(10, 10))
141     for i in range(n):
142         for j in range(n):
143             plt.subplot(n, n, i * n + j + 1)

```

```

144     plt.imshow(generated_images[i * n + j], cmap="gray")
145     plt.axis("off")
146     plt.savefig(f"generated_images_{latent_dim}.pdf")
147     plt.show()
148
149 # Plot scatter plot of training data in latent space
150 cvae.plot_scatter(train_images, train_labels)

```

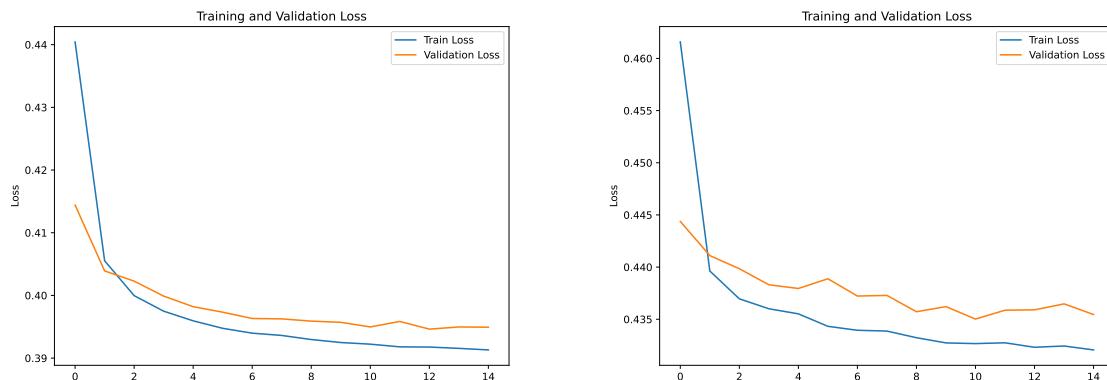
Program 13: Conv VAE + kNN Results

```

1 Latent Dimension: 32
2
3
4      precision    recall   f1-score   support
5
6          0       0.74      0.78      0.76      1000
7          1       0.90      0.83      0.86      1000
8          2       0.57      0.71      0.63      1000
9          3       0.65      0.79      0.71      1000
10         4       0.62      0.60      0.61      1000
11         5       0.81      0.86      0.84      1000
12         6       0.54      0.36      0.43      1000
13         7       0.82      0.79      0.80      1000
14         8       0.92      0.90      0.91      1000
15         9       0.90      0.87      0.89      1000
16
17 accuracy                  0.75      10000
18 macro avg                 0.75      0.75      10000
19 weighted avg               0.75      0.75      0.75      10000
20 -----
21
22 Latent Dimension: 64
23
24
25
26
27
28
29
30
31
32
33
34
35

```

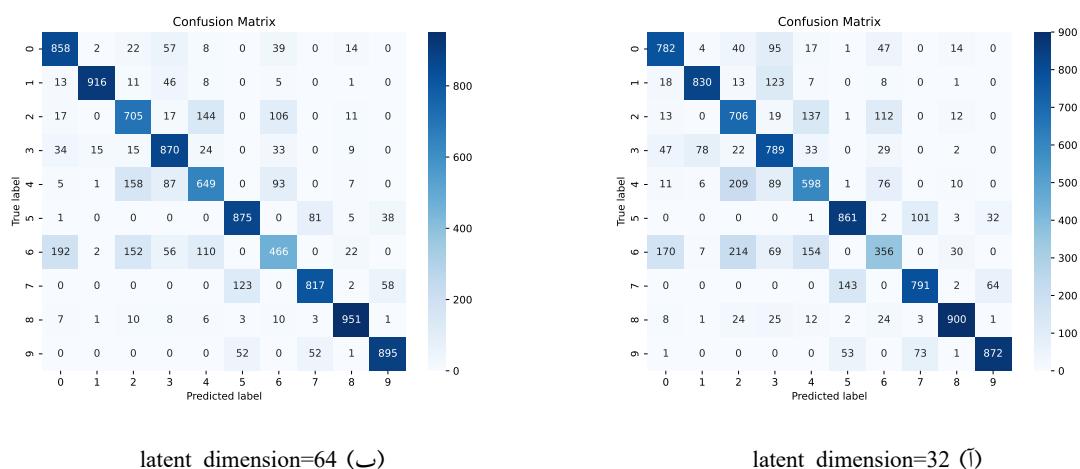
	precision	recall	f1-score	support
0	0.74	0.78	0.76	1000
1	0.90	0.83	0.86	1000
2	0.57	0.71	0.63	1000
3	0.65	0.79	0.71	1000
4	0.62	0.60	0.61	1000
5	0.81	0.86	0.84	1000
6	0.54	0.36	0.43	1000
7	0.82	0.79	0.80	1000
8	0.92	0.90	0.91	1000
9	0.90	0.87	0.89	1000
accuracy			0.75	10000
macro avg	0.75	0.75	0.75	10000
weighted avg	0.75	0.75	0.75	10000



(ب) latent_dimension=64

(ج) latent_dimension=32

شکل ۱۴: نمودار تابع اتلاف آموزش مدل خودرمزگذار متغیر کانولوشنی.



(ب) latent_dimension=64

(ج) latent_dimension=32

شکل ۱۵: نتیجه طبقه‌بندی در حالت خودرمزگذار متغیر کانولوشنی.

۵.۱ پاسخ قسمت ۵ - کاوش در فضای latent

همان‌طور که در قسمت قبلی اشاره شد، در کلاس تعریف شده برای VAE دستوراتی برای کاوش ذر فضای نهان نیز نوشته شده است. متند `plot_scatter` برای این کار تعریف شده که ورودی‌های آن عبارتند از `x_train` که داده‌های آموزشی هستند و `labels` که برچسب‌های مربوط به داده‌های آموزشی است. این متند با استفاده از مدل انکودر داده‌های آموزشی را به فضای نهان نگاشت می‌دهد و سپس نمودار Scatter از داده‌ها را در فضای نهان رسم می‌کند. در واقع این متند نمودار پراکندگی داده‌های آموزش را در فضای کاهش یافته نشان می‌دهد. از داده‌های آموزش استفاده می‌کند تا نمایش کاهش یافته آن‌ها را با استفاده از شبکه کدگذار بدست آورد. سپس داده‌های آموزش را بر اساس ابعاد کاهش یافته رسم می‌کند و بر اساس برچسب‌ها رنگ‌آمیزی می‌کند. هم‌چنین انکودر و دیکودر مدل VAE ذخیره هم می‌شوند. هم‌چنین با استفاده از دیکودر مدل VAE، تصاویر جدیدی تولید می‌شود. ابتدا یک نمونه از نویز نرمال با ابعاد مناسب تولید می‌شود و سپس با استفاده از دیکودر به تصاویر جدید تبدیل

می‌شود. نتایج در متغیر generated_images ذخیره می‌شود و به شکل یک شبکه تصاویر به اندازه ۱۰ در ۱۰ نمایش داده می‌شود. با ارائه این توضیحات دستورات مربوطه در [برنامه ۱۴](#) و [نتایج در شکل ۱۶](#) و [شکل ۱۷](#) آورده شده است. نتایج واضح‌تر و به‌فرمت پی‌دی‌اف در [این پیوند](#) آورده شده است و از آوردن‌شان در این گزارش بهدلیل افزایش حجم فایل اجتناب شده است.

Program 14: Latent VAE + kNN Results

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import tensorflow as tf
6 from tensorflow import keras
7 from tensorflow.keras import layers
8 from sklearn.model_selection import train_test_split
9 from sklearn.neighbors import KNeighborsClassifier
10 from sklearn.metrics import classification_report, confusion_matrix
11 from sklearn.preprocessing import MinMaxScaler
12 import warnings
13 warnings.filterwarnings('ignore')
14
15 # Load the fashion MNIST dataset
16 fashion_mnist = keras.datasets.fashion_mnist
17 (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
18
19 # Preprocess the data
20 scaler = MinMaxScaler()
21 train_images = train_images.reshape(train_images.shape[0], 28, 28, 1) / 255.0
22 test_images = test_images.reshape(test_images.shape[0], 28, 28, 1) / 255.0
23
24 # Define the CVAE class
25 class CVAE:
26     def __init__(self, input_shape, batch_size, latent_dim):
27         self.input_shape = input_shape
28         self.batch_size = batch_size
29         self.latent_dim = latent_dim
30
31         # Encoder
32         inputs = keras.Input(shape=self.input_shape)
33         x = layers.Conv2D(filters=32, kernel_size=3, strides=2, activation='relu', padding='same')(inputs)
34         x = layers.Conv2D(filters=64, kernel_size=3, strides=2, activation='relu', padding='same')(x)
35         x = layers.Flatten()(x)
36         x = layers.Dense(256, activation='relu')(x)
37         z_mean = layers.Dense(self.latent_dim, name="z_mean")(x)
38         z_log_var = layers.Dense(self.latent_dim, name="z_log_var")(x)

```

```

39
40     # Sampling layer
41
42     def sampling(args):
43         z_mean, z_log_var = args
44         epsilon = tf.random.normal(shape=(tf.shape(z_mean)[0], self.latent_dim))
45         return z_mean + tf.exp(0.5 * z_log_var) * epsilon
46
47
48     # Decoder
49
50     decoder_inputs = layers.Input(shape=(self.latent_dim,))
51     x = layers.Dense(7 * 7 * 64, activation='relu')(decoder_inputs)
52     x = layers.Reshape((7, 7, 64))(x)
53     x = layers.Conv2DTranspose(filters=64, kernel_size=3, strides=2, activation='relu',
54                               padding='same')(x)
55     x = layers.Conv2DTranspose(filters=32, kernel_size=3, strides=2, activation='relu',
56                               padding='same')(x)
57     outputs = layers.Conv2DTranspose(filters=1, kernel_size=3, activation='sigmoid', padding=
58                                     'same')(x)
59
60
61     # Models
62
63     self.encoder = keras.Model(inputs, z_mean)
64     self.decoder = keras.Model(decoder_inputs, outputs)
65     self.vae = keras.Model(inputs, self.decoder(self.encoder(inputs)))
66
67
68     # Loss function
69
70     reconstruction_loss = tf.reduce_mean(
71         keras.losses.binary_crossentropy(inputs, self.decoder(z)), axis=[1, 2]
72     )
73
74     kl_loss = -0.5 * tf.reduce_mean(1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var),
75                                     axis=-1)
76
77     vae_loss = tf.reduce_mean(reconstruction_loss + kl_loss)
78     self.vae.add_loss(vae_loss)
79     self.vae.compile(optimizer='adam')
80
81
82     def fit(self, x_train, epochs):
83
84         history = self.vae.fit(x_train, x_train, epochs=epochs, batch_size=self.batch_size,
85                               validation_split=0.1)
86         self.plot_loss(history)
87
88
89     def transform(self, x):
90
91         return self.encoder.predict(x)
92
93
94     def plot_loss(self, history):
95
96         plt.figure(figsize=(8, 6))

```

```

79     plt.plot(history.history['loss'], label='Train Loss')
80     plt.plot(history.history['val_loss'], label='Validation Loss')
81     plt.xlabel('Epochs')
82     plt.ylabel('Loss')
83     plt.title('Training and Validation Loss')
84     plt.legend()
85     plt.savefig(f'loss_plot_{self.latent_dim}.pdf')
86     plt.show()
87
88     def plot_scatter(self, x_train, labels):
89         latent_space = self.transform(x_train)
90         plt.figure(figsize=(8, 6))
91         sns.scatterplot(x=latent_space[:, 0], y=latent_space[:, 1], hue=labels, palette='tab10')
92         plt.xlabel('Latent Dimension 1')
93         plt.ylabel('Latent Dimension 2')
94         plt.title('Scatter Plot of Training Data in Latent Space')
95         plt.savefig(f'scatter_plot_{self.latent_dim}.pdf')
96         plt.show()
97
98 # Task 1: Train CVAE and classify reduced dimension space with KNN
99 latent_dims = [32, 64]
100
101 for latent_dim in latent_dims:
102     # Create CVAE instance
103     cvae = CVAE(input_shape=(28, 28, 1), batch_size=32, latent_dim=latent_dim)
104
105     # Train CVAE
106     cvae.fit(train_images, epochs=15)
107
108     # Reduce dimensionality of test data
109     reduced_test_images = cvae.transform(test_images)
110
111     # Classify reduced dimension space with KNN
112     knn = KNeighborsClassifier(n_neighbors=10)
113     knn.fit(reduced_test_images, test_labels)
114     knn_pred = knn.predict(reduced_test_images)
115
116     # Print classification report
117     print(f"Latent Dimension: {latent_dim}")
118     print(classification_report(test_labels, knn_pred))
119
120     # Plot confusion matrix
121     cm = confusion_matrix(test_labels, knn_pred)
122     df_cm = pd.DataFrame(cm, index=np.arange(10), columns=np.arange(10))
123     plt.figure(figsize=(8, 6))

```

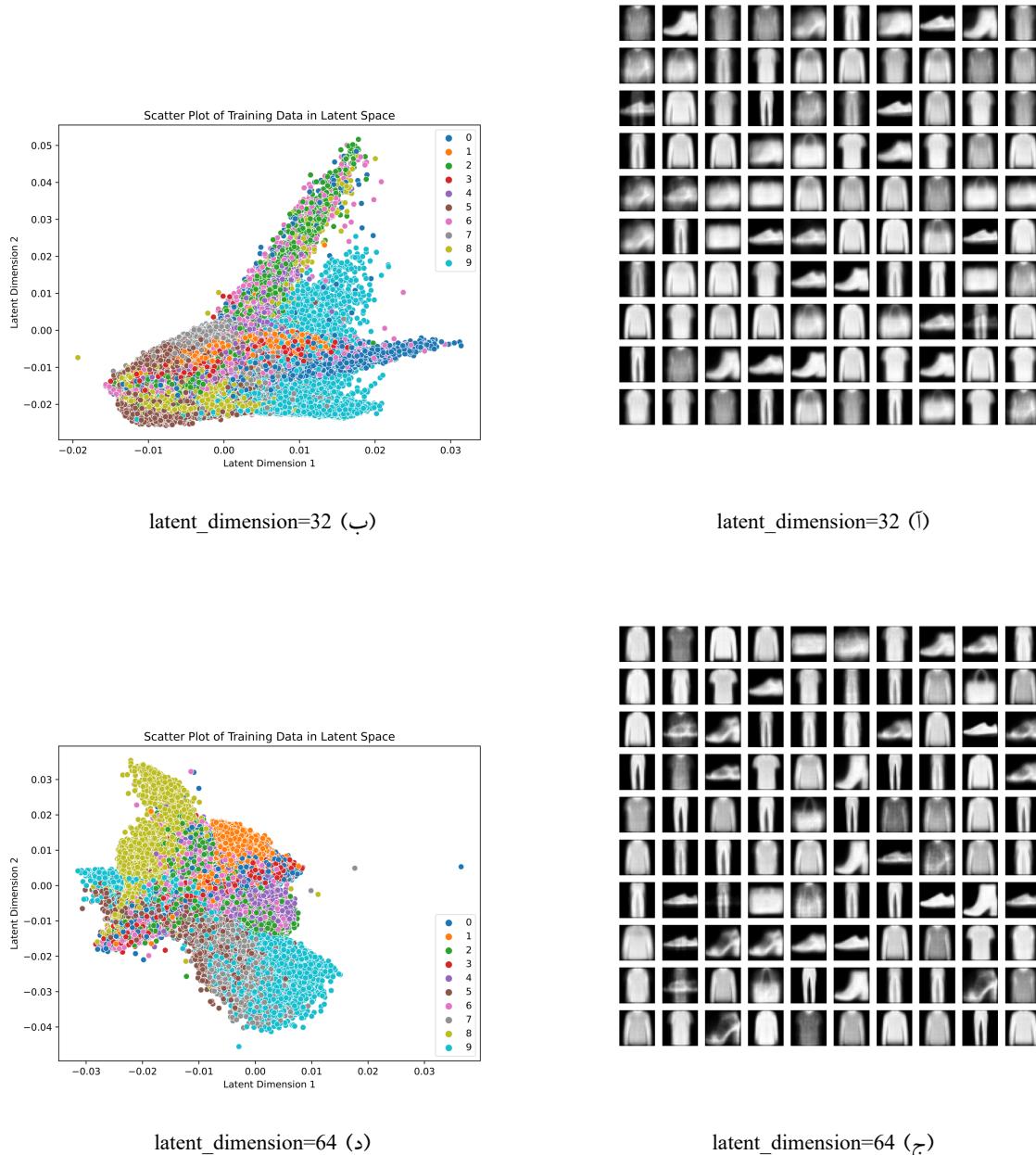
```

124     sns.heatmap(df_cm, annot=True, fmt="d", cmap="Blues")
125     plt.xlabel("Predicted label")
126     plt.ylabel("True label")
127     plt.title("Confusion Matrix")
128     plt.savefig(f"confusion_matrix_{latent_dim}.pdf")
129     plt.show()
130
131 # Save CVAE encoder and decoder networks
132 cvae.encoder.save(f"encoder_{latent_dim}.h5")
133 cvae.decoder.save(f"decoder_{latent_dim}.h5")
134
135 # Generate images using the decoder
136 n = 10
137 latent_samples = np.random.normal(size=(n * n, latent_dim))
138 generated_images = cvae.decoder.predict(latent_samples)
139 generated_images = generated_images.reshape(-1, 28, 28)
140 plt.figure(figsize=(10, 10))
141 for i in range(n):
142     for j in range(n):
143         plt.subplot(n, n, i * n + j + 1)
144         plt.imshow(generated_images[i * n + j], cmap="gray")
145         plt.axis("off")
146 plt.savefig(f"generated_images_{latent_dim}.pdf")
147 plt.show()
148
149 # Plot scatter plot of training data in latent space
150 cvae.plot_scatter(train_images, train_labels)

```

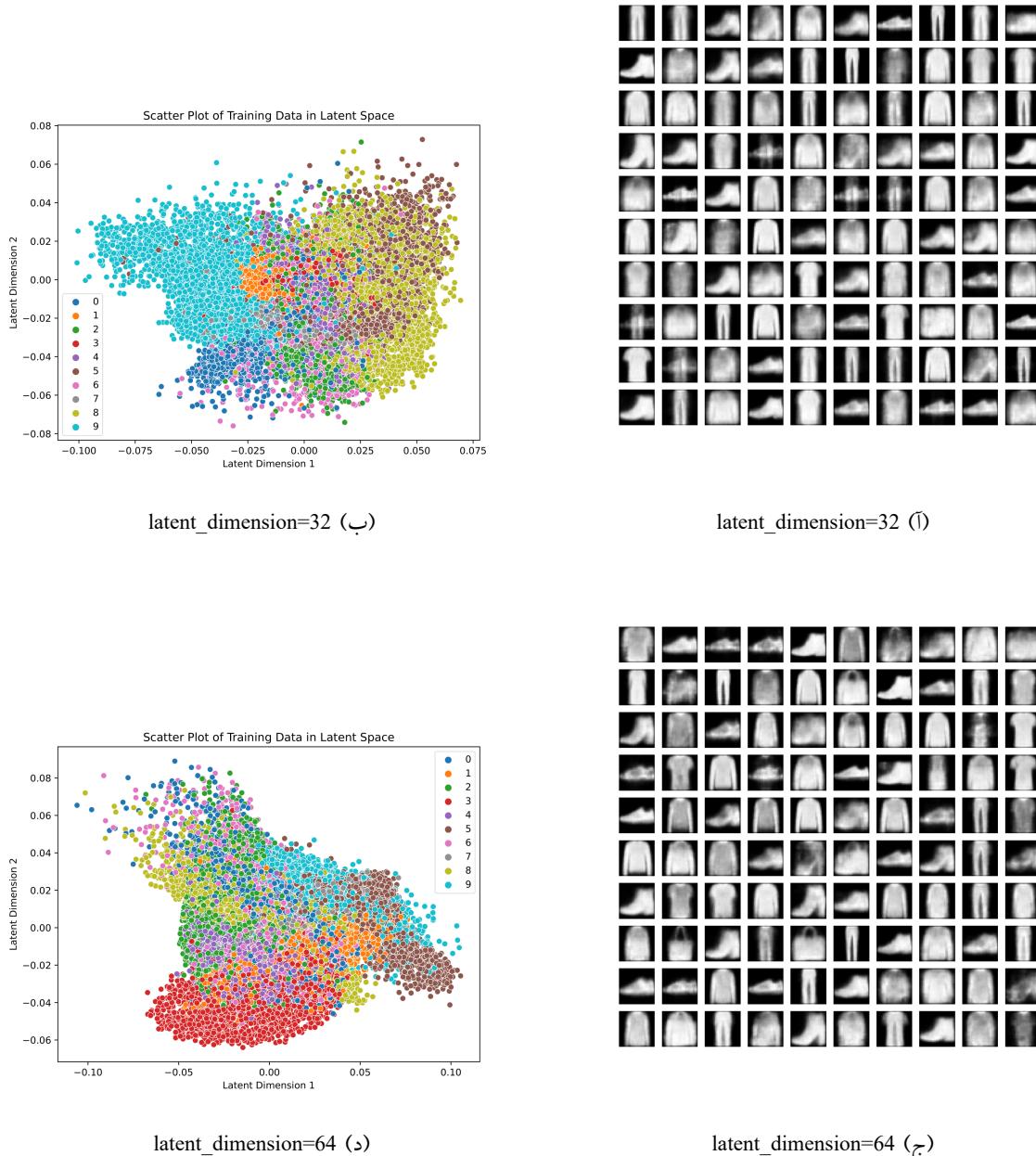
۶.۱ پاسخ قسمت ۶ - Diffusion Models

مدل‌های دفیوژن از یک فرآیند انتشار پیشرو ثابت و یک فرآیند انتشار معکوس قابل یادگیری تشکیل شده‌اند. فرآیند انتشار پیشرو یک فرآیند چند مرحله‌ای است که تدریج‌اً مقداری نویز گوسی کوچک را به نمونه اضافه می‌کند تا تا زمانی که به نویز سفید تبدیل شود. مقدار معمول برای تعداد مراحل این فرآیند ۱۰۰۰ است. فرآیند انتشار معکوس نیز یک فرآیند چند مرحله‌ای است که فرآیند انتشار پیشرو را معکوس می‌کند و نویز سفید را به تصویر برمی‌گرداند. هر مرحله از فرآیند انتشار معکوس توسط یک شبکه عصبی انجام می‌شود و تعداد مراحل آن با فرآیند پیشرو برابر است. فرآیند آموزش با بیشینه‌سازی لگاریتم احتمال، که پس از ساده‌سازی‌های ریاضی، به تابع خطای L_2 تبدیل می‌شود، انجام می‌شود. در طول آموزش، با استفاده از یک فرمول برای یک مقادیر T تصادفی اقدام به محاسبه تصاویر نویزی برای T و $T-1$ مرحله می‌کیم. سپس مدل دفیوژن، تصویر $T-1$ مرحله را از تصویر نویزی T مرحله پیش‌بینی می‌کند. تصویر تولیدشده و تصویر $T-1$ مرحله با استفاده از تابع خطای L_2 مقایسه می‌شوند. نمونه‌های با کیفیت بالای این مدل به دلیل خاصیت تدریجی حذف نویز است. برخلاف VAE و GAN که نمونه‌ها را به صورت یکجا تولید می‌کنند، مدل‌های دفیوژن نمونه‌ها را گام به گام ایجاد می‌کنند. این مدل ابتدا یک ساختار تصویر مبهم و خشن را ایجاد می‌کند و سپس بر روی آن جزئیات ریز را اضافه می‌کند. نمونه‌های با تنوع بالا از مزایای دیگر این مدل است.



شکل ۱۶: کاوش در فضای نهان مدل خودرمزگذار متغیر تمام‌امتصل.

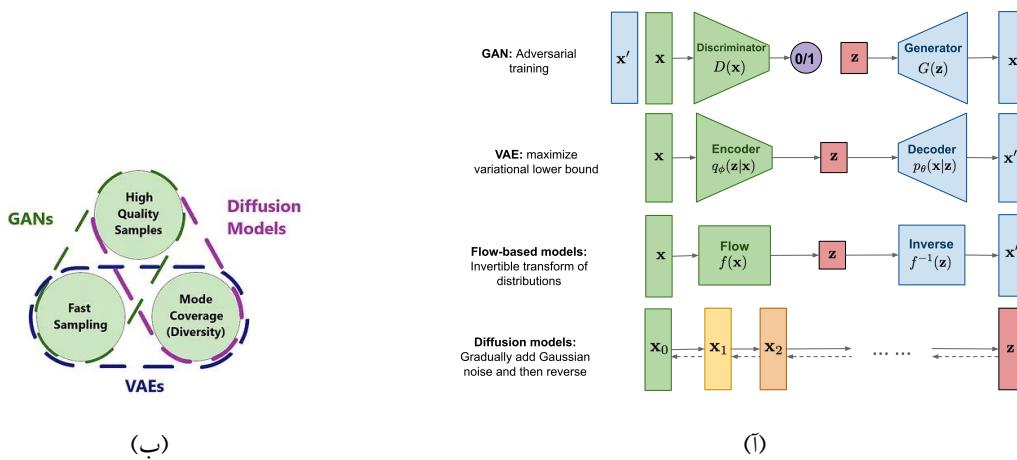
بیشینه‌سازی احتمال شامل تمام حالت‌های موجود در مجموعه داده آموزشی می‌شود. تصاویر نویزی میانی به عنوان کدهای نهان عمل می‌کنند و دارای همان اندازه تصاویر آموزشی هستند. این یکی از دلایلی است که مدل‌های دفیوژن قادر به تولید نمونه‌های با کیفیت بالا هستند. آموزش آسانی هم دارند. آن‌ها دارای یکتابع خطای قابل محاسبه برای بیشینه‌سازی احتمال هستند. این مدل‌ها برخلاف های GAN و VAE نیاز به چندین بار اجرای شبکه عصبی برای تولید نمونه‌ها به صورت تدریجی دارد. اگرچه روش‌های نمونه‌برداری وجود دارند که می‌توانند این فرآیند را به طور قابل توجهی سریع‌تر کنند، اما همچنان از های GAN و VAE به طور قابل توجهی کنتر هستند. فرآیند چند مرحله‌ای، این مدل امکانات جدیدی مانند پرکردن نقاط



شکل ۱۷: کاوش در فضای نهان خودرمزگذار متغیر کانولوشنی.

خالی (inpainting) یا تولید تصویر به تصویر، به سادگی با بهره برداری از نویز ورودی فراهم می‌کند. فرآیند پیش روی این مدل‌ها با نمونه داده‌های نمونه برداری شده از توزیع واقعی داده‌ها آغاز می‌شود و در آن مقدار کوچکی از نویز گوسی به نمونه اضافه می‌شود و به صورت مرحله‌ای، دنباله‌ای از نمونه‌های نویزی تولید می‌شود. اندازه مراحل توسط برنامه زمانی واریانس کنترل می‌شود.

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N} \left(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I} \right) \quad q(\mathbf{x}_{1:T} | \mathbf{x}_0) = \prod_{t=1}^T q(\mathbf{x}_t | \mathbf{x}_{t-1}) \quad (3)$$



شکل ۱۸: مقایسه مدل‌های مختلف شبکه‌های مولد.

یک ویژگی خوب این فرایند این است که می‌توانیم به راحتی در هر مرحله زمانی دلخواه در یک فرم بسته از ترفند بازتعریف نمونه‌برداری استفاده کنیم:

$$\begin{aligned} \mathbf{x}_t &= \sqrt{\alpha_t} \mathbf{x}_{t-1} + \sqrt{1 - \alpha_t} \epsilon_{t-1} \\ &= \sqrt{\alpha_t \alpha_{t-1}} \mathbf{x}_{t-2} + \sqrt{1 - \alpha_t \alpha_{t-1}} \bar{\epsilon}_{t-2} \\ &= \dots \\ &= \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon \end{aligned}$$

$$q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I})$$

where $\epsilon_{t-1}, \epsilon_{t-2}, \dots \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\bar{\epsilon}_{t-2}$ merges two Gaussians (*).

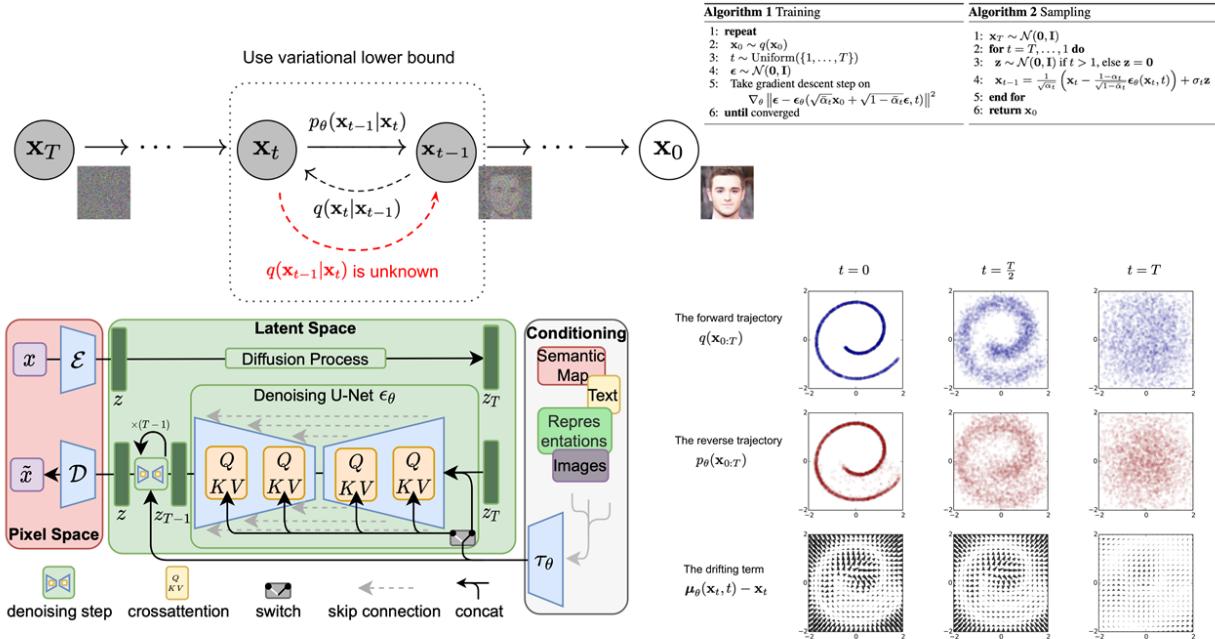
به خاطر داشته باشید که وقتی دو توزیع گوسی با واریانس‌های متفاوت، و، را ترکیب می‌کنیم، توزیع جدید به شکل یک توزیع گوسی با میانگین مشترک و واریانس مشترک خواهد بود. در اینجا، واریانس ترکیب شده برابر است با:

$$\sqrt{(1 - \alpha_t) + \alpha_t (1 - \alpha_{t-1})} = \sqrt{1 - \alpha_t \alpha_{t-1}} \quad (4)$$

عموماً می‌توانیم در هنگامی که نمونه بیشتر نویز دارد، قدم بروزرسانی بزرگتری بگذاریم. اگر بتوانیم فرایند فوق را معکوس کنیم و از توزیع نمونه برداری کنیم، قادر خواهیم بود نمونه واقعی را از ورودی نویز گوسی، بازسازی کنیم.

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) \quad p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \quad (5)$$

بنابراین، مدل‌های انتشار یا Diffusion Models به عنوان یک روش یادگیری تولید مولد استفاده می‌شوند. این مدل‌ها از یک روش آماری برای ایجاد داده‌های جدید بر اساس داده‌های آموزشی موجود استفاده می‌کنند. اصل این مدل‌ها بر اساس فرآیند انتشار تکانه است که با انتشار گام به گام در طول زمان، توزیع احتمالی برای داده‌های جدید را تولید می‌کند. عملکرد اصلی مدل‌های انتشار این است که با استفاده از چندین مرحله انتشار، از یک توزیع احتمالی ساده شروع کرده و به تدریج به یک توزیع پیچیده‌تر می‌رسد. هر مرحله انتشار یک فرآیند تکانه‌ای است که با انتخاب نقطه‌ای از توزیع جاری، نقطه جدیدی با توزیع جدیدی را تولید می‌کند. این عملکرد تکانه‌ای در مدل‌های انتشار، امکان تولید داده‌های جدید و نوآورانه را فراهم می‌کند. مدل‌های انتشار با وجود کمبودهایی مانند پیچیدگی محاسباتی بالا و وابستگی به ترتیب زمانی، توانسته‌اند نتایج بسیار موفقی در



.Diffusion Models :۱۹

تولید داده‌های با کیفیت بالا و تصاویر ویدیویی واقع‌گرایانه ارائه دهنده. این مدل‌ها معمولاً بر روی داده‌هایی با ساختار پیچیده مانند تصاویر و صدای استفاده می‌شوند. در مقابل، مدل‌های Variational Autoencoders (VAEs) یک ساختار متفاوت برای یادگیری تولید داده دارند. VAEs از یک شبکه عصبی کدگشا و کدگذار استفاده می‌کنند. این مدل‌ها با یادگیری توزیع احتمالی مقدار نهان، (latent)، امکان تولید داده‌های جدید را دارند. مزیت مدل‌های انتشار نسبت به VAEs در این است که می‌توانند و با نمونه‌برداری از این توزیع، داده‌های جدید را تولید کنند. مزیت مدل‌های انتشار نسبت به VAEs در این است که می‌توانند توزیع‌های پیچیده‌تری را نمایش دهند و در نتیجه تولید داده‌های با کیفیت بالاتری انجام دهند. اما این مدل‌ها دارای هزینه محاسباتی بالایی هستند و نیاز به زمان زیادی برای آموزش دارند. همچنین، وابستگی به ترتیب زمانی می‌تواند محدودیت‌هایی را برای استفاده از این مدل‌ها در بعضی موارد اعمال کند. به طور کلی، مدل‌های انتشار به دلیل قابلیت تولید داده‌های با کیفیت بالا و نوآورانه، در برخی از حوزه‌ها مانند تولید تصاویر و ویدیوهای کاربرد دارند. در مقابل، VAEs به دلیل ساختار ساده‌تر و هزینه محاسباتی کمتر، در بسیاری از برنامه‌های ماشین و تولید داده مورد استفاده قرار می‌گیرند.