به نام خدا



گزارش پروژه شبکه­های عصبی

درس هوش محاسباتی

دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

استاد درس : دکتر حسین کارشناس

تهیه کنندگان :  
علیرضا دستمالچی ساعی، محمد حسین دهقانی، محمد توکلی

اردیبهشت 1402

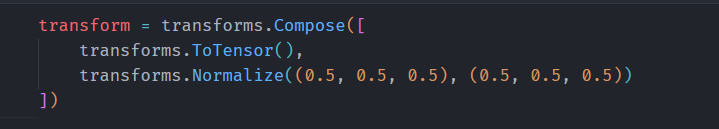
**مقدمه :**

در این پروژه قصد داریم تعدادی تصویر را دسته­بندی کنیم. یکی از رویکردهای مناسب جهت دسته­بندی تصاویر، استفاده از شبکه­های عصبی می­باشد. در قسمت اول این پروژه ابتدا دسته بندی تصاویر توسط شبکه عصبی Resnet34 انجام می­شود که معیار ما برای سنجش درستی شبکه عصبی آموزش دیده شده توسط خودمان است. گام بعدی پس از انتخاب معماری شبکه، انتخاب وزن های درست برای شبکه می­باشد که در قسمت اول پروژه نیز، هدف پیدا کردن همین وزن­ها برای دسته بندی هرچه بهتر تصاویر می­باشد.

**روش انجام :**

در فاز ابتدایی پروژه ابتدا نیاز داریم که دیتاست هایی را برای تمرین و ارزیابی مدل حاصل از این قسمت آماده‌سازی کنیم. برای این کار از زبان پایتون و کتابخانه torchvision استفاده می‌کنیم. در این کتابخانه ماژول‌های models, datasets و trasforms و ماژول DataLoader از کتابخانه torch استفاده می‌کنیم.

پیاده سازی این قسمت به صورت زیر انجام می‌شود.

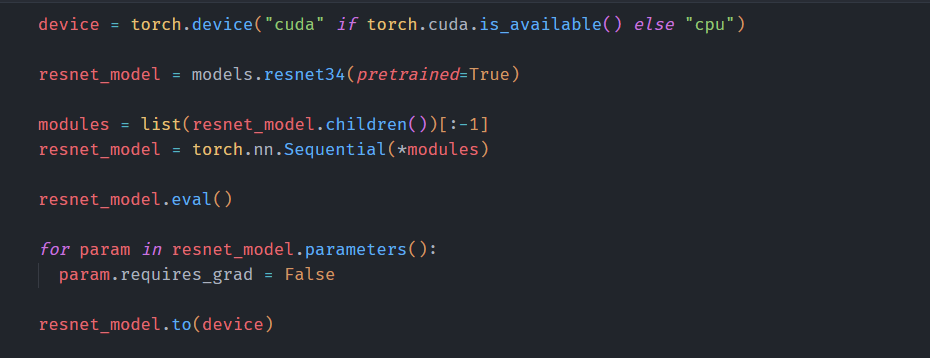


**A picture containing text, screenshot, font

Description automatically generated**

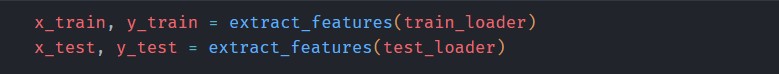
**پس از انجام مراحل بالا** train\_loader و test\_loader آماده استخراج ویژگی‌ها به وسیله شبکه resnet34 هستند و به ترتیب برای آموزش و ارزیابی مدل مورد استفاده قرار خواهند گرفت.

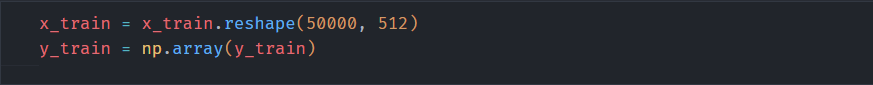
در گام بعدی باید با استفاده از resnet34 بردار ویژگی مدنظر را از داده‌ها استخراج کنیم. از آنجایی که این کار زمان نسبتا زیادی را برای اجرا شدن نیاز دارد از قدرت پردازشی gpu برای انجام محاسبات این بخش استفاده شده‌است. در این گام با حذف لایه آخر شبکه resnet34 بردار ویژگی‌های مدنظر را از داده‌ها استخراج می‌کنیم. مراحل تعریف resnet34، حذف آخرین لایه این شبکه، جلوگیری از آموزش دیدن این شبکه و نهایتا انتقال آن به gpu در تصویر زیر قابل مشاهده است.



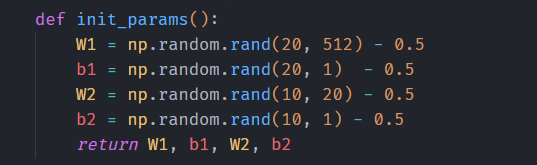
در گام بعدی نوبت به استخراج ویژگی‌ها با استفاده از resnet34 می‌رسد. تصاویر زیر نحوه انجام این کار را نشان می‌دهند.



****

****

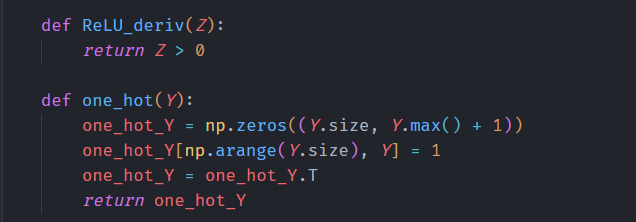
**پس از استخراج ویژگی‌های مدنظر با استفاده از** resnet34 نوبت به پیاده سازی شبکه عصبی خودمان با ویژگی‌های خواسته شده می‌رسد. این شبکه دارای سه لایه می‌باشد که شامل یک لایه ورودی، یک لایه میانی (لایه پنهان) و یک لایه خروجی است. با توجه به ویژگی‌های شبکه عصبی و بردارهای ویژگی استخراج شده پارامترهای شبکه عصبی را مقداردهی اولیه می‌کنیم.



**سایر توابعی که برای پیاده سازی این شبکه عصبی نیاز داریم عبارتند از تابع فعال‌ساز** ReLU، تابع softmax، تابع one\_hot و ... می‌باشند که پیاده سازی آنها را در تصاویر زیر مشاهده می‌کنید.

A picture containing text, screenshot, font

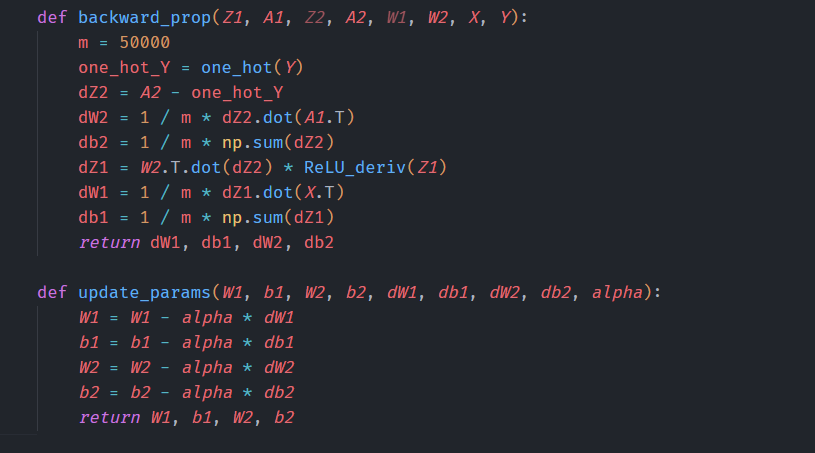
Description automatically generated

****

**علاوه بر توابع بالا به توابع** forward\_prop و backward\_prop برای پیمایش شبکه و به دست آوردن حاصل و اصلاح وزن های شبکه و تابع update\_params هم نیاز داریم که برای به‌روزرسانی پارامترهای شبکه کاربرد دارد. پیاده‌سازی این توابع را در تصاویر زیر مشاهده می‌کنید.

A picture containing text, screenshot, font, number

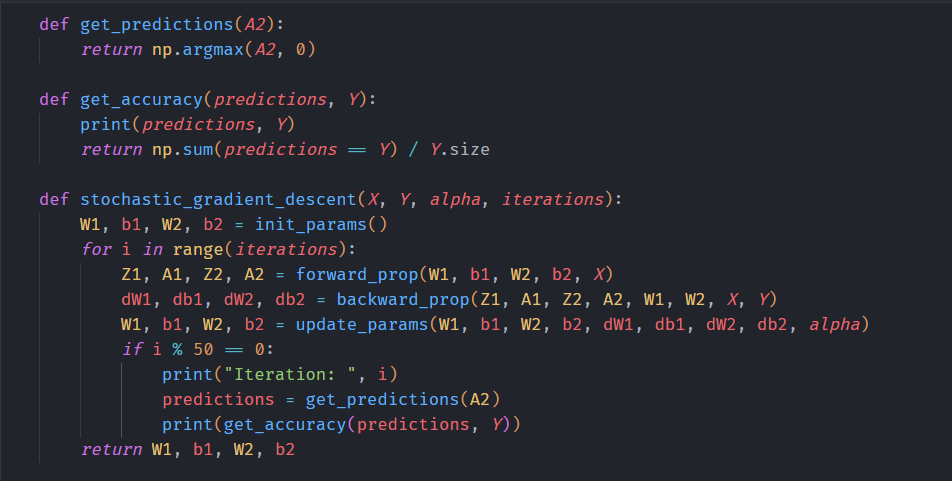
Description automatically generated

****

**پس از هربار پیمایش شبکه از لایه ورودی به سمت لایه خروجی و به دست آمدن خروجی و میزان خطای شبکه با استفاده از الگوریتم پس انتشار خطا اقدام به به‌روزرسانی وزن‌های شبکه می‌کنیم تا با گذشت هر مرحله خطای به دست آمده کمتر و کمتر شود و مدلی با دقت بالا داشته باشیم.**

**در نهایت با استفاده از تابع** SGD **به آموزش و ارزیابی شبکه می‌پردازیم. این تابع با ترکیب توابع تعریف شده قبلی ابتدا شبکه را آموزش داده و پس از هر مرتبه به‌روزرسانی وزن‌ها و بایاس‌های شبکه دقت شبکه را مورد ارزیابی قرار می‌دهد و خروجی ها را به فرمت مناسبی نمایش می‌دهد.**

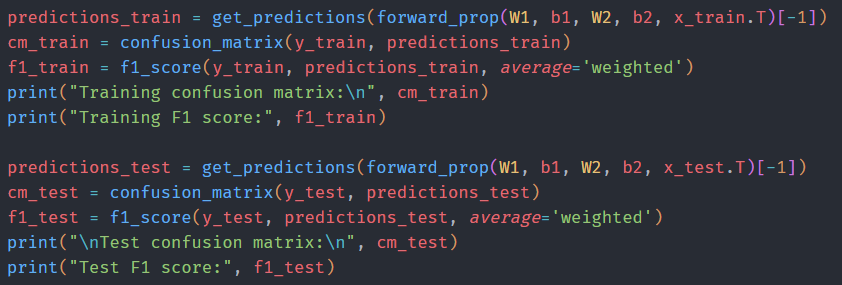
**پیاده سازی تابع** SGD و سایر توابع مربوطه را در تصویر زیر مشاهده می‌کنید.

****

**فراخوانی تابع** SGD بر روی داده‌های تمرینی با ضریب یادگیری و تعداد دور مشخص.

****

**در پایان فاز اول پروژه ماتریس درهم‌ریختگی و امتیاز** F1 مدل آموزش داده شده توسط قطعه کد زیر محاسبه می‌شوند.



**نتایج اجرا برای 20 دور و ضریب یادگیری داده شده به این شرح است:**

Iteration: 0

[5 4 3 ... 5 6 1] [6 9 9 ... 3 7 7]

Accuracy: 7.201999999999999 %

Iteration: 1

[5 2 4 ... 5 6 1] [6 9 9 ... 3 7 7]

Accuracy: 8.584 %

Iteration: 2

[9 2 4 ... 5 6 1] [6 9 9 ... 3 7 7]

Accuracy: 9.628 %

Iteration: 3

[9 2 4 ... 5 6 1] [6 9 9 ... 3 7 7]

Accuracy: 10.372 %

Iteration: 4

[9 2 4 ... 5 6 1] [6 9 9 ... 3 7 7]

Accuracy: 11.042 %

Iteration: 5

[9 2 4 ... 5 6 1] [6 9 9 ... 3 7 7]

Accuracy: 11.612 %

Iteration: 6

[9 2 4 ... 5 6 1] [6 9 9 ... 3 7 7]

Accuracy: 12.24 %

Iteration: 7

[9 2 4 ... 5 6 1] [6 9 9 ... 3 7 7]

Accuracy: 12.844 %

Iteration: 8

[9 2 4 ... 0 6 1] [6 9 9 ... 3 7 7]

Accuracy: 13.446 %

Iteration: 9

[9 2 4 ... 0 6 1] [6 9 9 ... 3 7 7]

Accuracy: 14.084 %

Iteration: 10

[9 2 4 ... 0 6 1] [6 9 9 ... 3 7 7]

Accuracy: 14.734 %

Iteration: 11

[9 2 4 ... 0 6 1] [6 9 9 ... 3 7 7]

Accuracy: 15.354 %

Iteration: 12

[9 2 4 ... 0 6 1] [6 9 9 ... 3 7 7]

Accuracy: 15.994 %

Iteration: 13

[9 2 4 ... 0 6 1] [6 9 9 ... 3 7 7]

Accuracy: 16.636 %

Iteration: 14

[9 2 4 ... 0 6 1] [6 9 9 ... 3 7 7]

Accuracy: 17.276 %

Iteration: 15

[9 2 4 ... 0 6 1] [6 9 9 ... 3 7 7]

Accuracy: 17.942 %

Iteration: 16

[9 2 4 ... 0 6 1] [6 9 9 ... 3 7 7]

Accuracy: 18.58 %

Iteration: 17

[9 2 4 ... 0 6 1] [6 9 9 ... 3 7 7]

Accuracy: 19.28 %

Iteration: 18

[9 2 4 ... 0 6 1] [6 9 9 ... 3 7 7]

Accuracy: 19.95 %

Iteration: 19

[9 2 4 ... 0 6 1] [6 9 9 ... 3 7 7]

Accuracy: 20.62 %

Training confusion matrix:

[[4010 61 235 105 106 102 68 101 653 459]

[ 204 4148 66 87 57 86 71 85 265 931]

[ 462 56 2746 439 685 445 358 392 64 53]

[ 134 68 558 2065 374 1602 330 436 41 32]

[ 181 40 569 256 3152 282 283 411 65 21]

[ 83 35 511 1033 271 3342 174 438 43 10]

[ 23 39 384 297 201 180 4131 92 26 7]

[ 97 33 284 156 318 321 47 3674 28 42]

[ 613 136 80 50 80 74 36 55 3899 987]

[ 277 454 64 60 44 62 29 77 240 3803]]

Training F1 score: 0.6153867036361499

Test confusion matrix:

[[810 18 90 27 26 26 17 37 231 138]

[ 49 889 14 20 8 19 13 25 62 901]

[131 16 600 63 160 102 71 110 16 17]

[ 33 22 174 439 90 340 70 103 11 8]

[ 38 10 167 62 635 76 52 100 16 4]

[ 17 18 143 348 77 620 36 112 14 5]

[ 6 15 121 90 48 47 858 23 9 3]

[ 28 12 87 41 107 95 15 741 11 35]

[177 54 30 18 31 29 10 18 582 51]

[ 80 152 17 14 10 22 7 26 48 634]]

Test F1 score: 0.4910593188355117

**تحلیل نتایج :**

**نتایج به دست آمده در مرحله قبل را می‌توان با افزایش تعداد دور به صورت قابل توجهی بهبود داد. به نتایج زیر که از اجرای همان الگوریتم و صرفا با افزایش تعداد دور به دست آمده‌اند توجه کنید:**

Iteration: 0

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 9.152 %

Iteration: 50

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 40.57000000000001 %

Iteration: 100

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 45.424 %

Iteration: 150

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 48.624 %

Iteration: 200

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 51.438 %

Iteration: 250

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 53.982 %

Iteration: 300

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 56.052 %

Iteration: 350

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 58.150000000000006 %

Iteration: 400

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 59.82000000000001 %

Iteration: 450

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 61.354 %

Iteration: 500

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 62.762 %

Iteration: 550

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 63.984 %

Iteration: 600

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 64.84000000000002 %

Iteration: 650

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 65.66000000000001 %

Iteration: 700

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 66.48 %

Iteration: 750

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 67.27000000000001 %

Iteration: 800

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 68.096 %

Iteration: 850

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 68.874 %

Iteration: 900

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 69.582 %

Iteration: 950

[6 5 4 ... 4 5 8] [6 9 9 ... 1 1 5]

Accuracy: 70.328 %

Training confusion matrix:

[[1969 0 4 14 6 15 26 4 103 17]

[ 3 2350 1 6 1 7 7 2 38 44]

[ 159 0 1422 77 120 72 105 73 23 5]

[ 45 3 19 1626 63 181 46 34 25 18]

[ 28 1 25 24 1907 28 58 99 13 3]

[ 34 2 12 134 48 1749 19 56 20 29]

[ 16 2 22 59 27 19 1843 8 4 0]

[ 11 0 10 34 64 28 2 1865 5 22]

[ 29 0 1 6 2 5 4 2 1959 18]

[ 11 5 0 9 1 7 4 7 36 1930]]

Training F1 score: 0.701220594044287

Test confusion matrix:

[[1263 49 182 75 133 155 149 130 504 360]

[ 160 838 145 120 143 152 158 73 469 902]

[ 357 21 358 266 378 395 281 215 92 37]

[ 180 29 159 361 168 786 184 119 63 51]

[ 129 16 179 163 674 383 150 248 62 16]

[ 102 22 130 384 134 871 128 135 46 8]

[ 60 55 146 205 246 174 1090 32 24 8]

[ 139 41 152 206 374 437 88 467 58 18]

[ 288 83 67 77 57 47 44 34 1085 276]

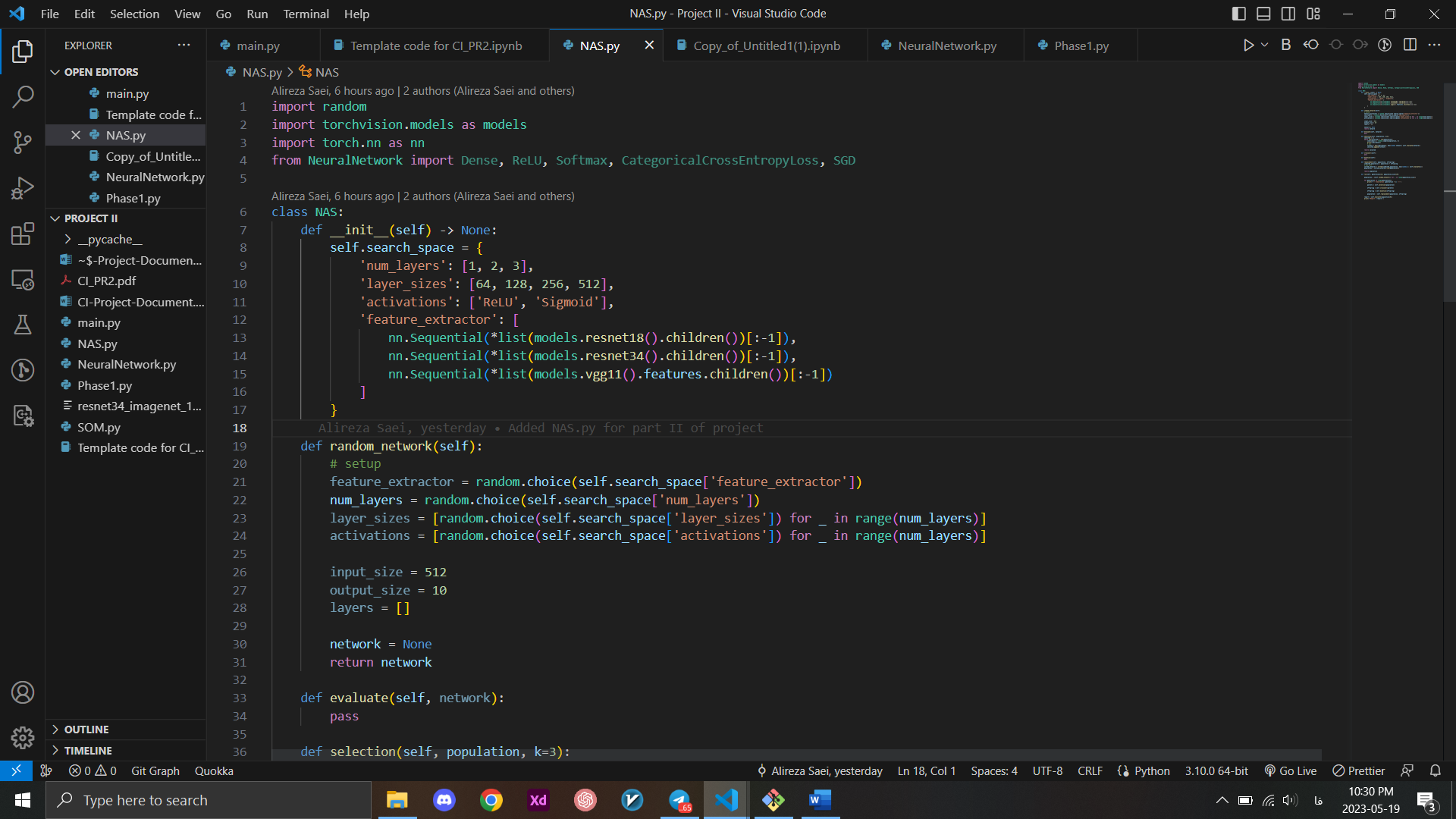
[ 193 147 51 106 45 112 60 73 191 1022]]

Test F1 score: 0.3086479529518663

جست و جوی معماری شبکه عصبی (NAS)

**در این بخش از الگوریتم­های تکاملی برای پیدا کردن بهترین معماری استفاده شده است. بدین صورت که با اندازه جمعیت 10 معماری به اندازه 10 نسل جلو رفته و درنهایت بهترین معماری با بهترین fitness را انتخاب می­کنیم.**

**کلاس NAS هنگام ساخته شدن مقادیر فضای جست و جو را در خود ذخیره می­کند:**

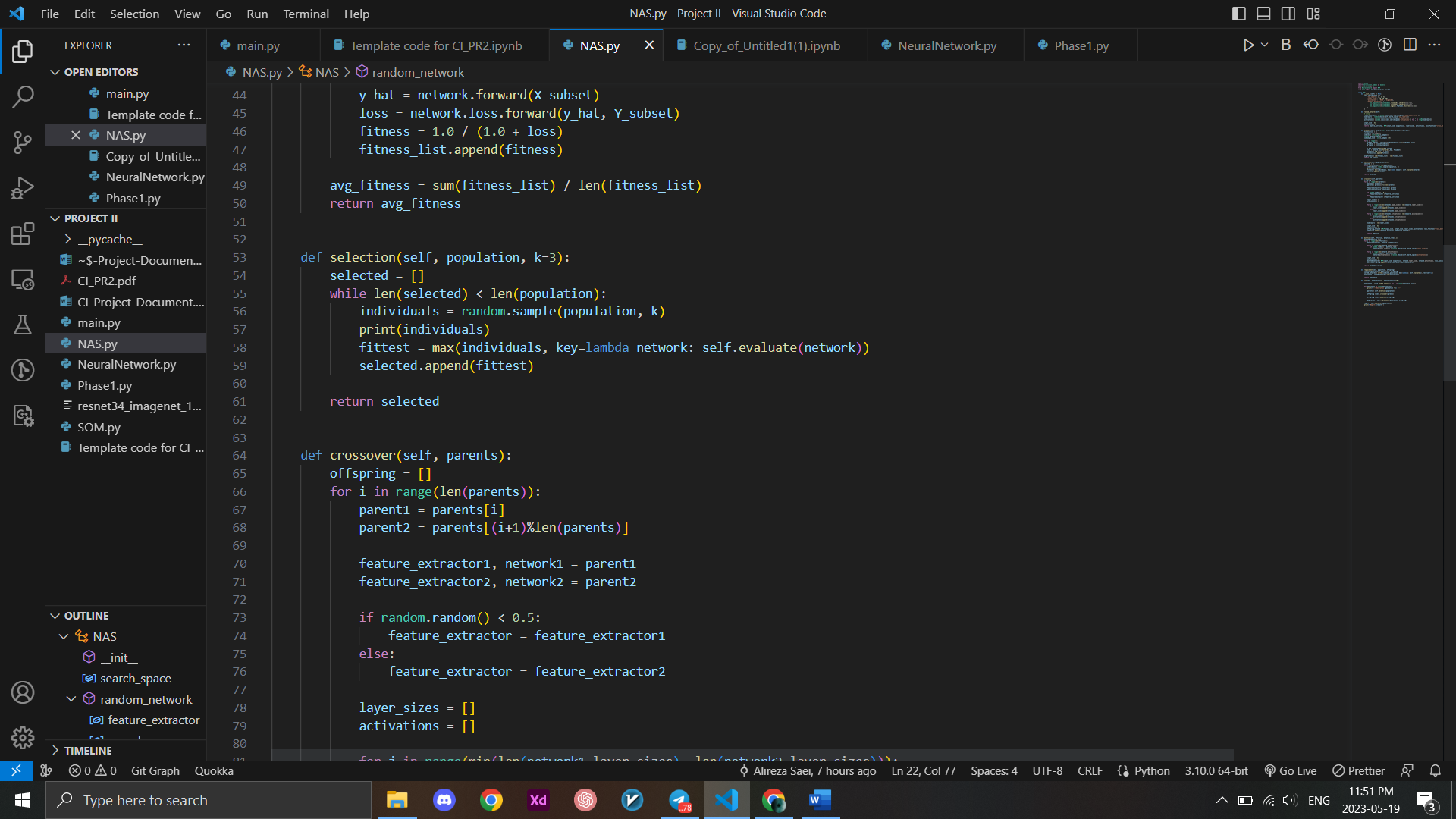


و هنگام اجرای الگوریتم تکاملی، عملگرهای تغییر برای تکامل از این مقادیر استفاده می­کنند. سپس برای مقداردهی اولیه از فضای جست و جو به صورت رندوم انتخاب می­کنیم توسط تابع random\_network که با صدا زدن آن یک معماری خروجی می­دهد.

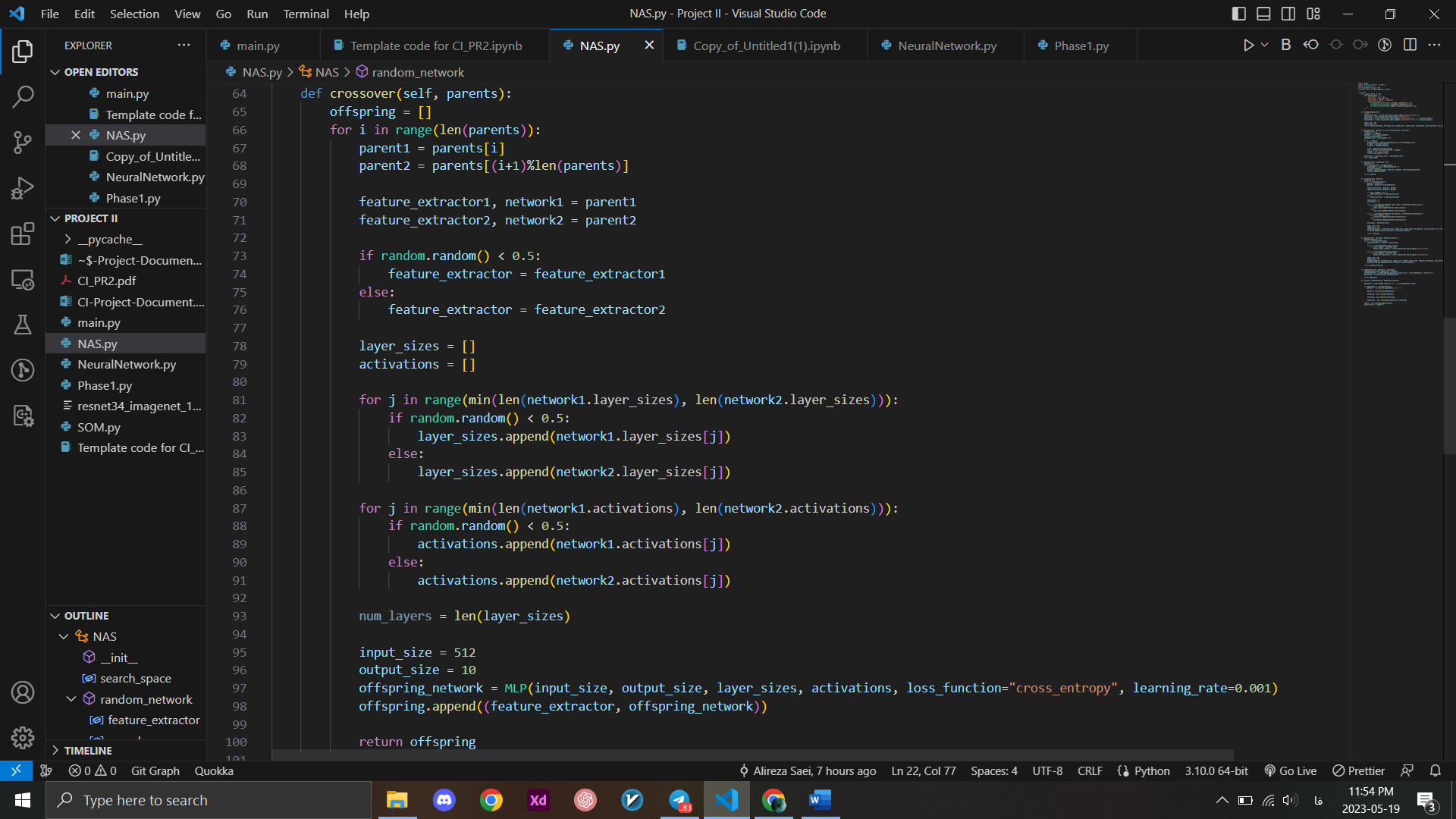
**ویژگی­های الگوریتم­ تکاملی استفاده شده**

برای الگوریتم تکاملی در فاز 2 پروژه، قسمت­های مختلف را به صورت زیر پیاده­سازی کرده­ایم:

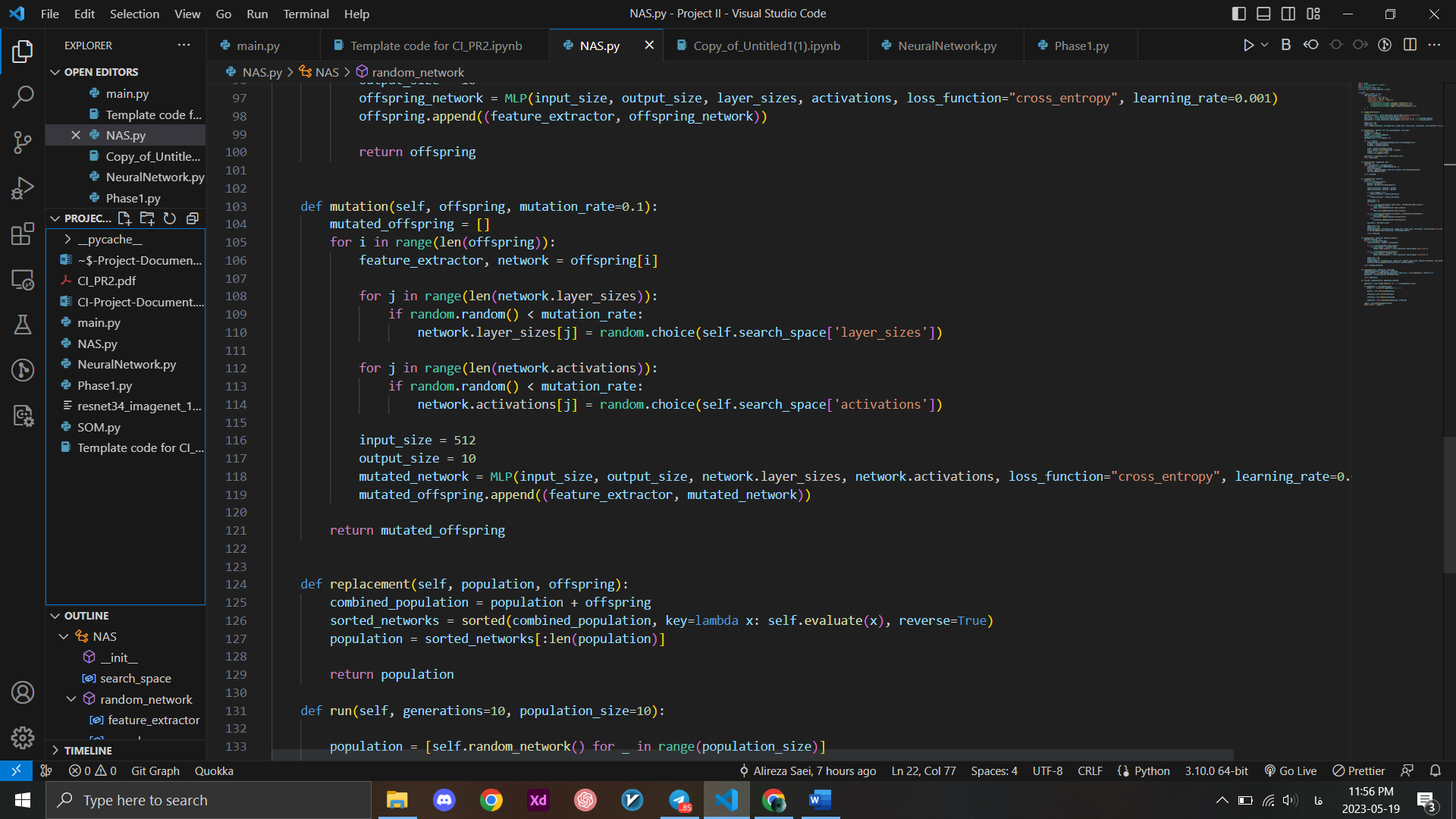
1. **انتخاب:** روش انتخاب مورد استفاده در الگوریتم تکاملی به صورت tournament selection می­باشد که هربار به صورت رندوم 3 شبکه را انتخاب و یکی از بهترین آن­ها را انتخاب می­کند.



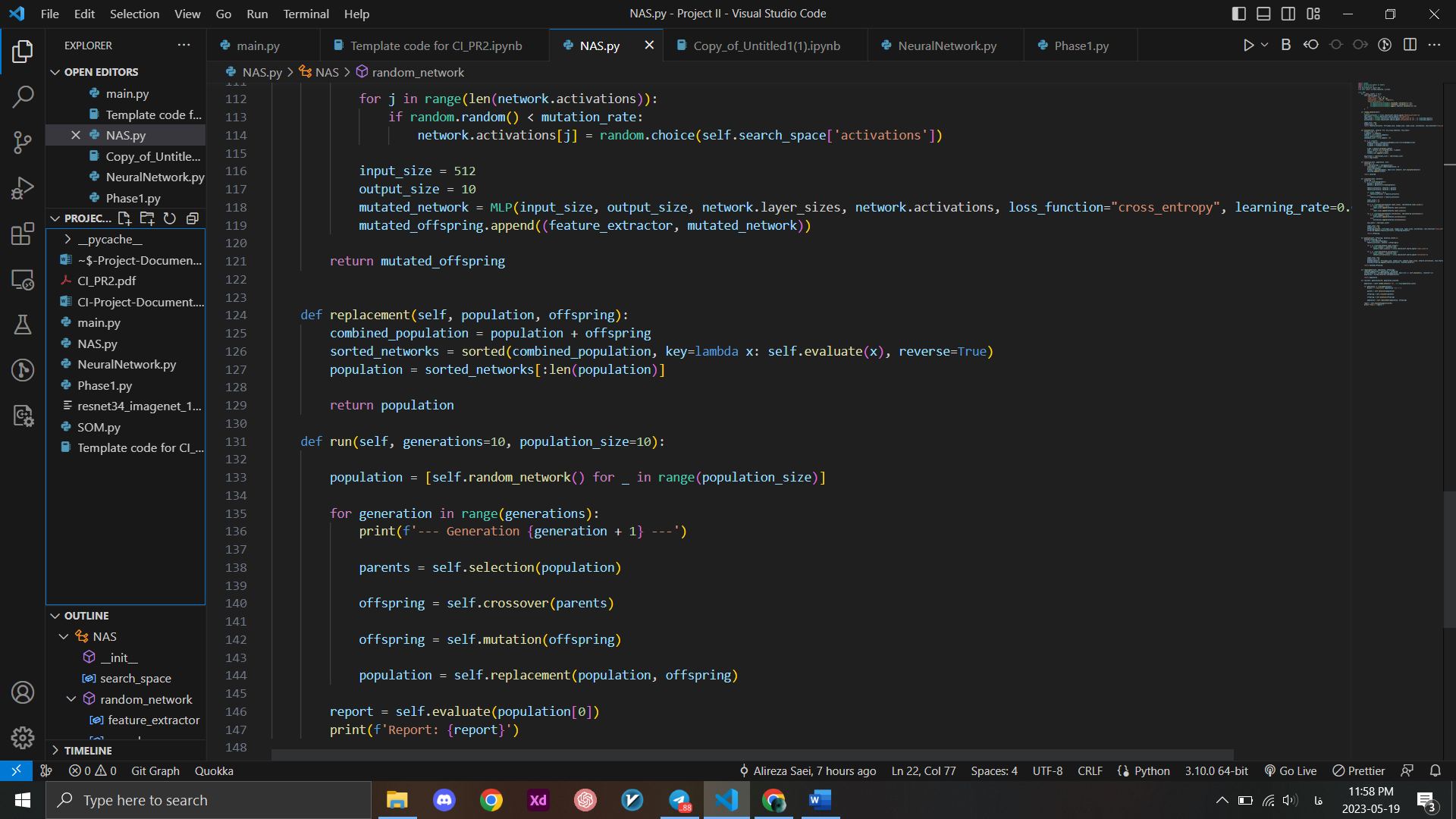
1. **بازترکیب:** روش بازترکیب به کار گرفته شده بدین صورت است که از crossover برا تغییر استخراج کننده ویژگی­ها و لایه­ها و توابع فعال­سازی آنها استفاده شده است. تمامی احتمال های انتخاب شده برای بازترکیب 0.5 می­باشد:



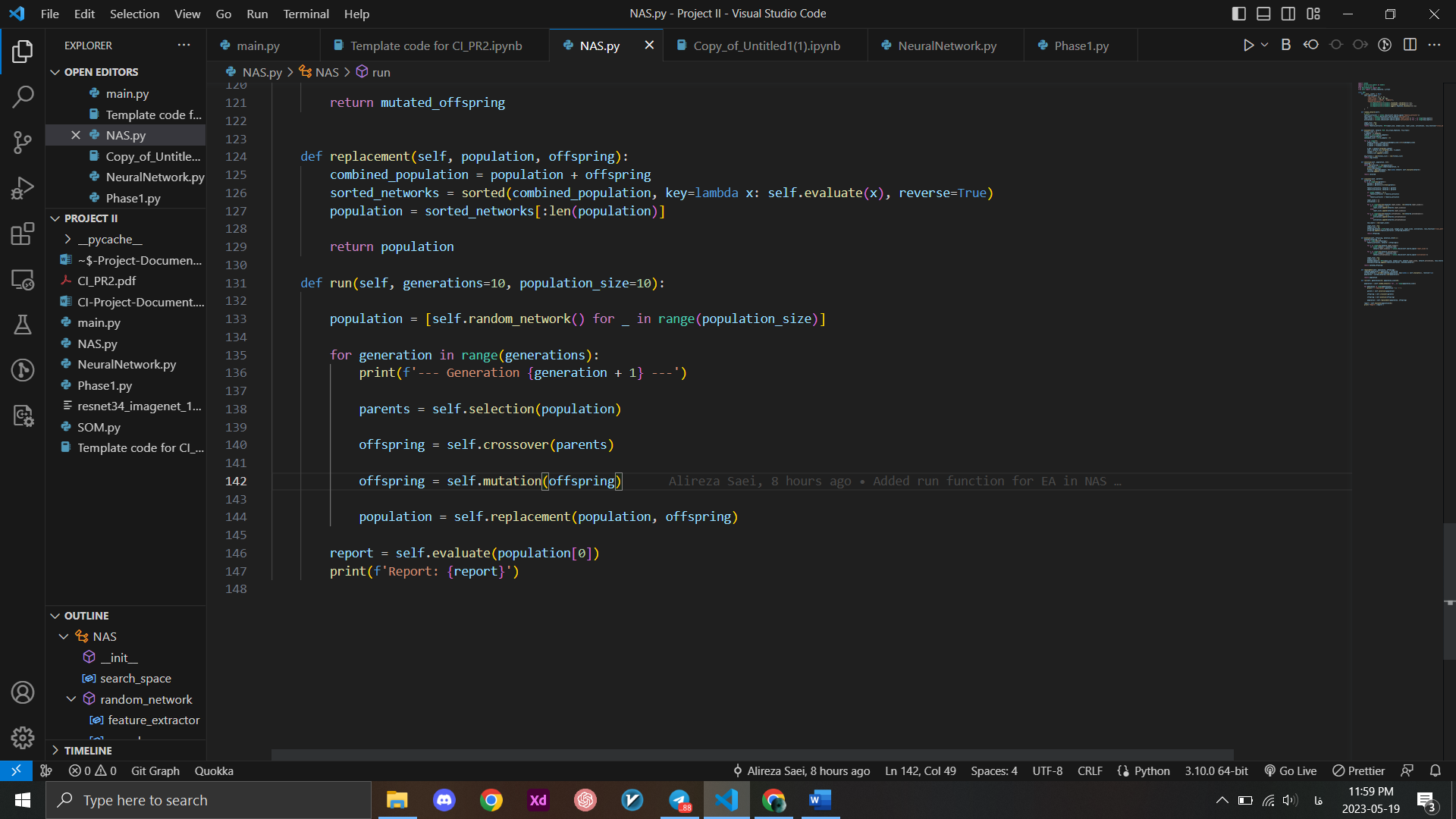
1. **جهش:** برای عملگر جهش نیز با احتمال 0.1 متغیرهای فضای جست و جو تغییر پیدا می­کند:



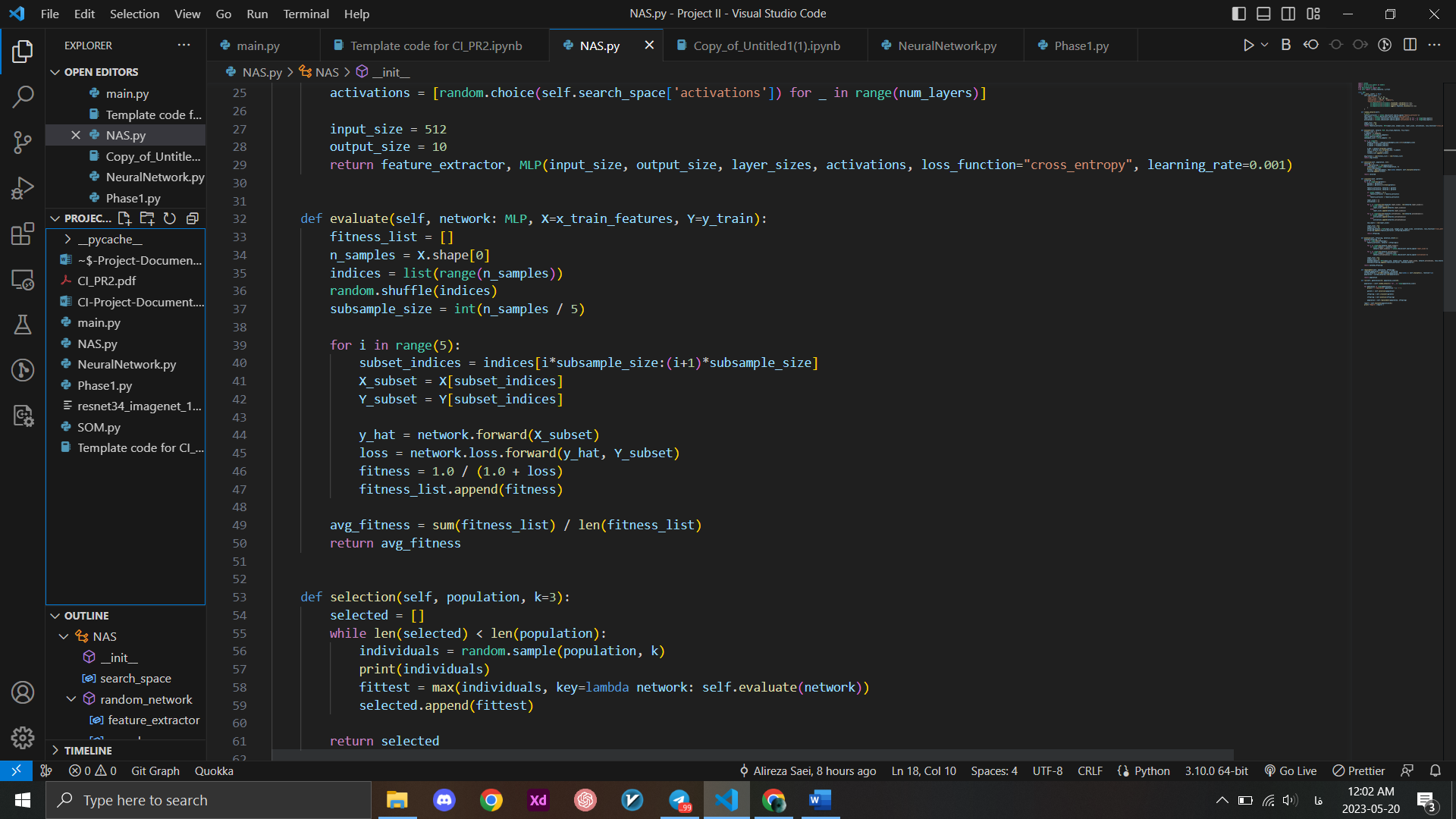
1. **جای­گذاری:** در عملیات جای­گذاری، از ساده­ترین روش استفاده شده و بدین گونه است که ابتدا شبکه­ها براساس fitness خود مرتب شده، سپس از میان آنها به تعداد اندازه جمعیت بهترین آنهارا انتخاب می­کنیم:



1. **اجرا:** این تابع الگوریتم تکاملی را با توجه به آرگومان­های پاس داده شده به تابع اجرا می­کند. که عبارتند از تعداد نسل­ها و تعداد جمعیت. این تابع پس از اتمام اجرا، نتیجه­ای از عملکرد خود را چاپ می­کند:



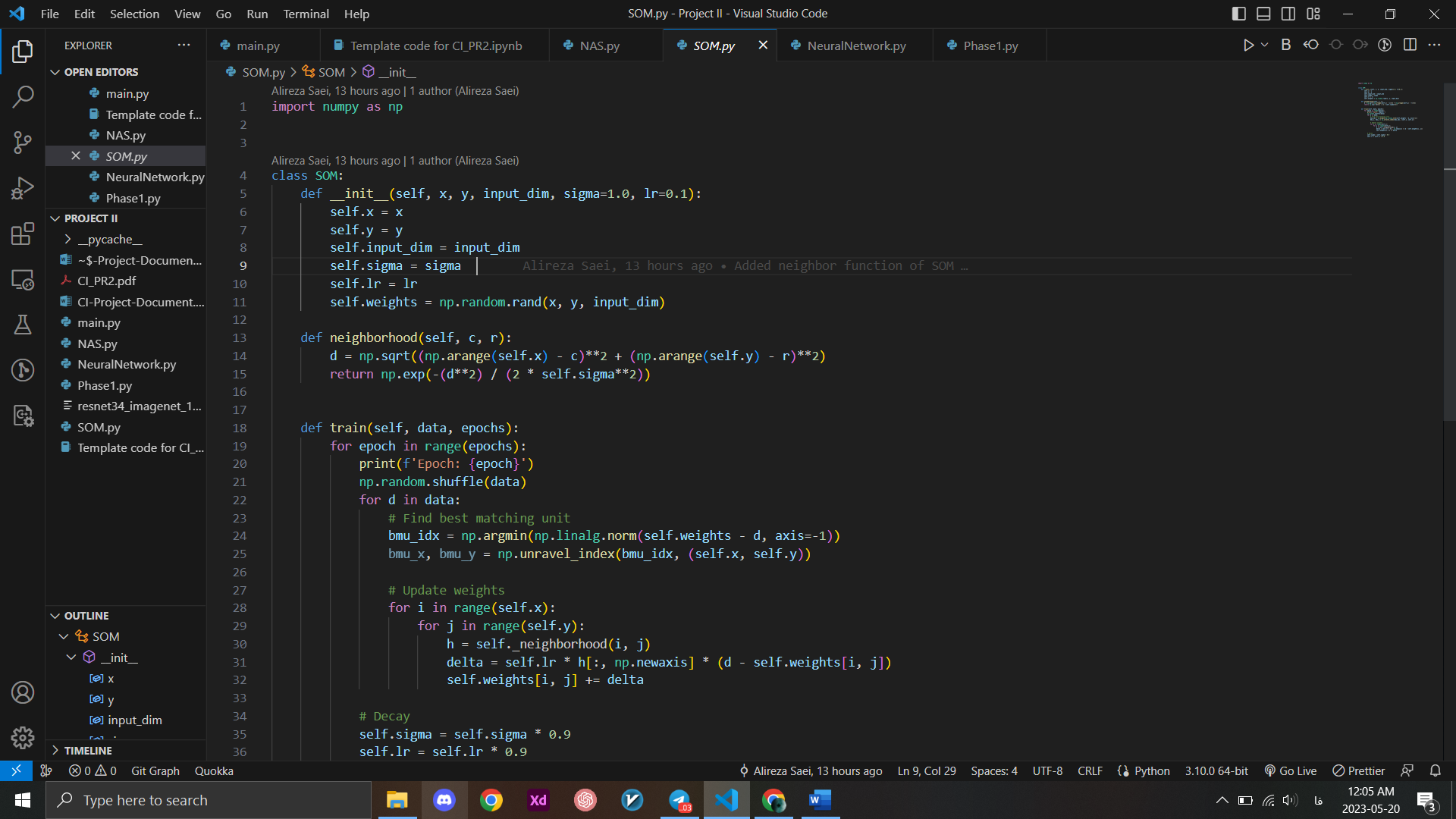
1. **برازندگی:** درنهایت به قسمت Evaluation الگوریتم تکاملی خود می­رسیم که این تابع با گرفتن مقادیر ورودی و خروجی و اجرا به اندازه 5 بار با لیست­های مختلف برای بدست آوردن برازندگی، نتیجه نهایی را به صورت میانگین برازندگی هر 5 بار ارسال می­کند:



**تحلیل نتایج**  
با اجرای الگوریتم تکاملی و انتخاب درست هایپرپارامترها و ... معماری­های انتخاب شده به سوی دقت بالاتر و loss کمتر حرکت می­کند و در نهایت معماری­ای با بهترین استخراج کننده ویژگی، بهترین تعداد لایه­ها و بهترین تعداد نورون هر لایه همراه با توتاع فعال­سازی آنها. بدلیل مدت زمان اجرای بلند نمودارها پس اجرا شدن روی سرورهای گوگل قرار گرفته می­شوند.

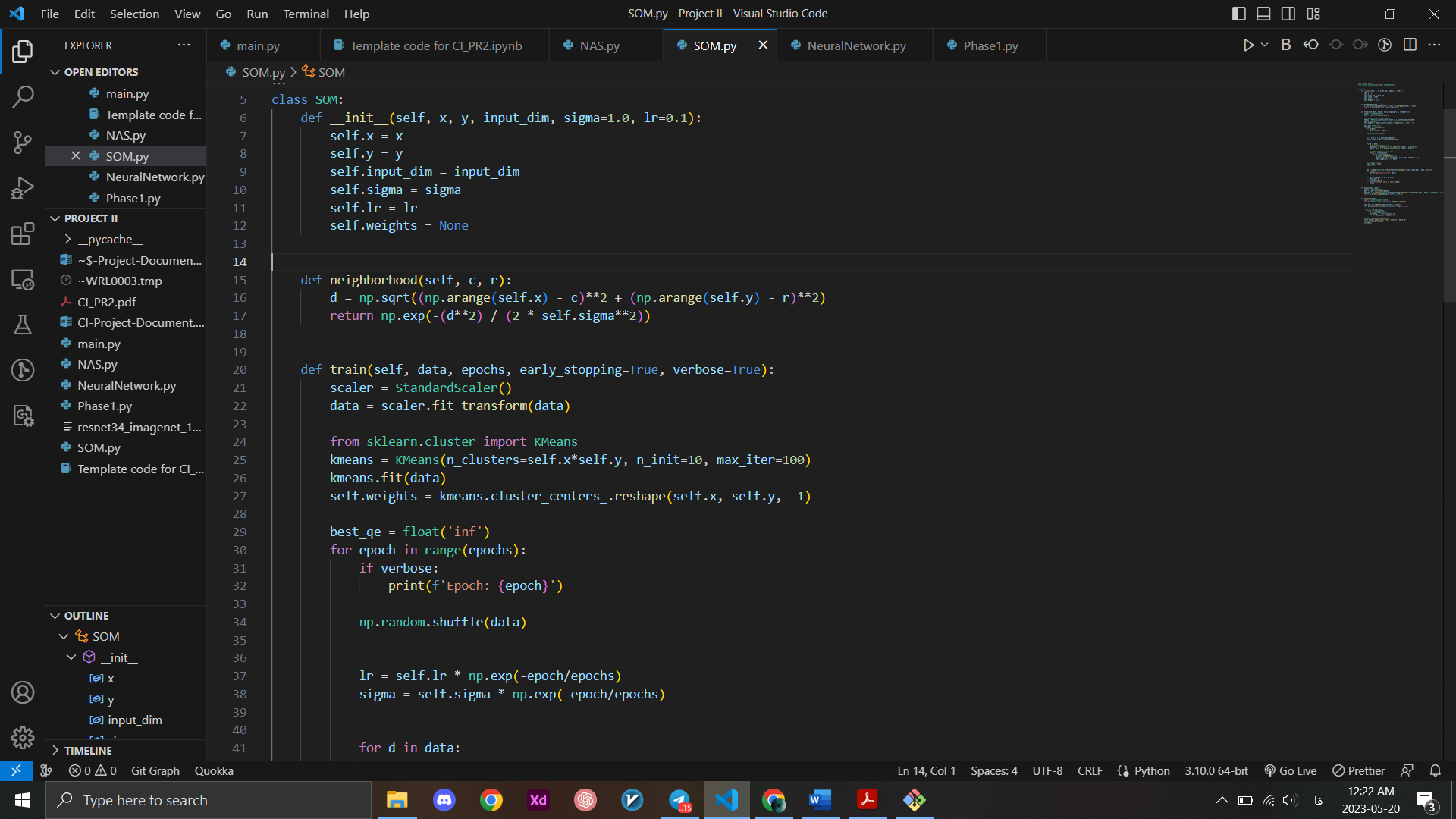
**خوشه­بندی عکس­ها با SOM**

در این بخش، بدون درنظر گرفتن لیبل­های دیتاست و صرفا براساس ویژگی­های بدست آمده، اقدادم به خوشه بندی می­کنیم. برای این کار از SOM استفاده شده است. بدین صورت که کلاس SOM به صورت زیر مقداردهی می­شود (موقع initialize کردن):



سپس از توابع دیگر که برای خوشه بندی استفاده شده اند:

neighborhood اول تابع همسایگی گاوسی را با مرکزیت مختصات (c,r) محاسبه می کند. این تابع در تابع دوم، Train(self, data, epochs) استفاده می شود که الگوریتم (SOM) را برای یادگیری نمایش داده های با ابعاد بالا با ابعاد پایین پیاده سازی می کند.



تابع "train" با استفاده از داده ورودی، شبکه‌ی SOM را برای تعداد مشخصی از دوره‌ها آموزش می‌دهد. در ادامه جزئیات عملکرد این تابع آمده است:

1. داده ورودی با استفاده از StandardScaler نرمال سازی می‌شود.

2. وزن‌های شبکه با استفاده از الگوریتم K-Means مقداردهی اولیه می‌شوند.

3. برای هر دوره از آموزش، مراحل زیر اجرا می‌شود:

- داده‌های ورودی را به صورت تصادفی مخلوط می‌کنیم.

- نرخ یادگیری و فاصله همسایگی براساس شماره فعلی دوره محاسبه می‌شوند.

- برای هر داده ورودی، واحدی که بیشترین شباهت را با آن دارد (BMU) پیدا می‌شود.

- وزن واحدهای شبکه بر اساس فاصله آن‌ها از BMU و همچنین فاصله همسایگی با BMU بروزرسانی می‌شوند.

- نرخ یادگیری و فاصله همسایگی با توجه به شماره فعلی دوره، کاهش می‌یابد.

- خطای کوانتیزاسیون (QE) محاسبه می‌شود. این مقدار میانگین فاصله بین هر داده ورودی و BMU آن است.

- در صورت ارتباط با معیار متوقف کننده، آموزش در صورت لزوم متوقف می‌شود.

4. SOM آموزش داده شده، با استفاده از تابع "predict" قابل استفاده است تا بتوانیم BMUs برای داده‌های جدیدی پیش‌بینی کنیم.

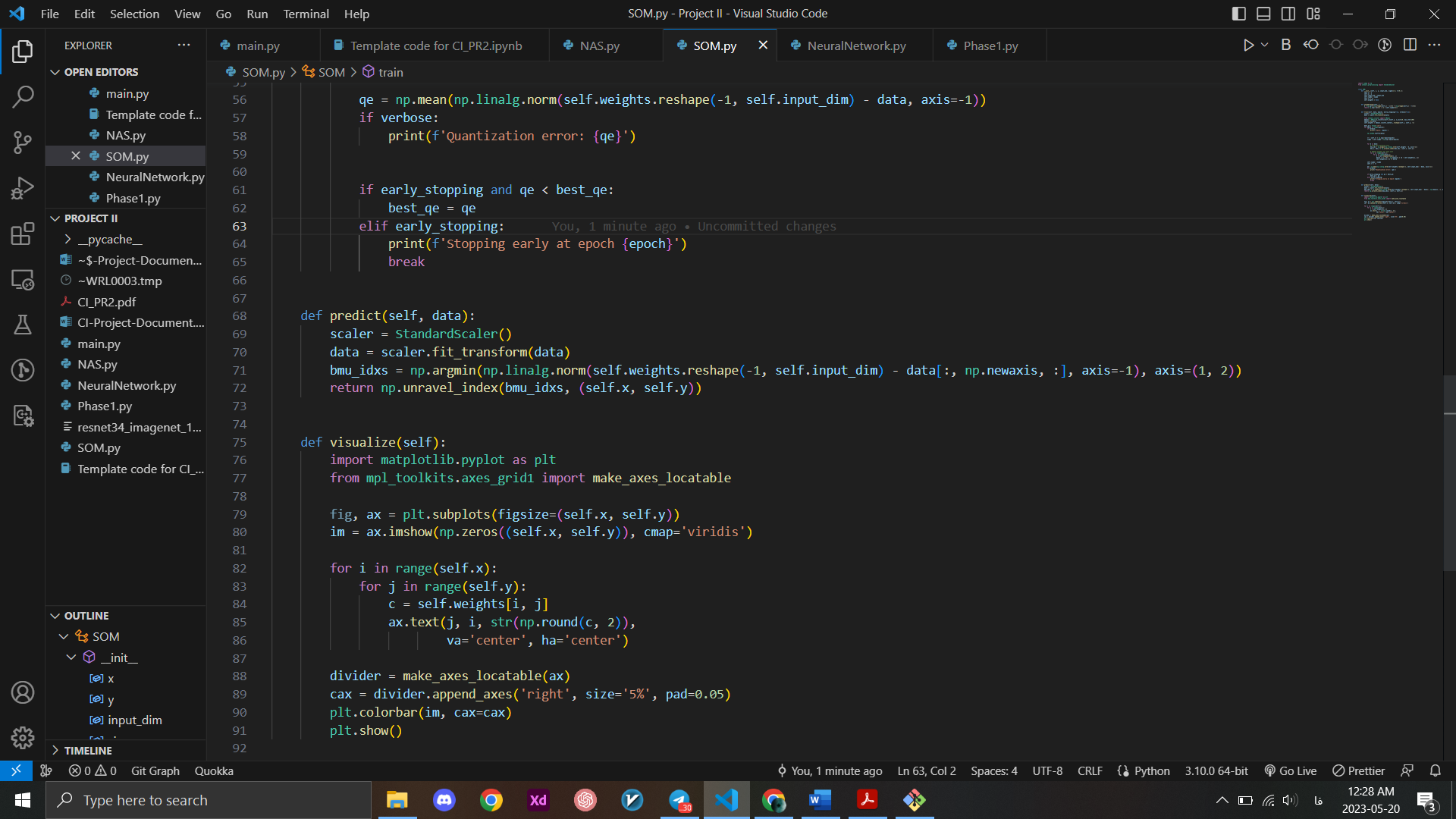
5. تابع "visualize" می‌تواند برای رسم شبکه SOM با برچسب‌های متنی و رنگ‌آمیزی استفاده شود.

در کل، تابع "train" وزن‌های SOM را به‌گونه‌ای بروزرسانی می‌دهد که داده‌های مشابه در همسایگی یکدیگر در شبکه SOM قرار گیرند. با این کار، نگاشتی از فضای بعد بالای داده ورودی به یک شبکه دو بعدی (یا چند بعدی) کوچکتر به‌وجود می‌آید که به تصویر سازی و تحلیل الگوهای داده امکان می‌دهد.

تابع "predict" در کلاس SOM یک مجموعه از داده‌های ورودی را گرفته و به ازای هر داده بهترین واحد تطبیق (BMU) را برمی‌گرداند.

در ابتدا، داده‌های ورودی با استفاده از `StandardScaler` نرمال‌سازی می‌شوند. سپس برای هر داده ورودی، BMU را با محاسبه فاصله اقلیدسی بین آن و تمام واحدهای شبکه SOM پیدا می‌کند. BMU به عنوان واحدی تعریف می‌شود که دارای کمترین فاصله اقلیدسی با داده ورودی است.

تابع "predict" مختصات BMU را برای هر داده ورودی در قالب یک تاپل با دو آرایه (یکی برای مختصات x و دیگری برای مختصات y) برمی‌گرداند. اندازه این آرایه‌ها متناظر با تعداد داده‌های ورودی است.



**تقسیم وظایف:**

* **علیرضا دستمالچی ساعی**

فایل­های main و SOM و Neural Network و NAS و داک مربوط به بخش های 2و3 داک

* **محمدحسین دهقانی**

فایل­های phase1 و داک مربوط به بخش 1 داک

* **محمدتوکلی**

قالب­بندی کلاس­ها و تمپلیت داک و رفع باگ­های runtime در کدها