



یادگیری ماشین - بهار ۱۴۰۴

گزارش نهایی پروژه

امیر محمد عالمیان (۸۱۰۱۰۲۱۹۸)

شهریار رحیمی راد (۸۱۰۱۰۲۱۵۱)

سید احمد حیدریه (۸۱۰۱۰۳۱۰۹)

## بخش اول

### معماری بکار رفته

در این بخش مدلی بر مبنای faster R-CNN پیاده‌سازی شده است که شامل اجزای backbone, Anchor generator, RegionProposalNetwork و RoiHeads می‌باشد. در بخش اول از یک معماری شبیه ResNet18 که توسط خودمان و با الهام گیری از ResNet18 نوشته شده، برای قسمت backbone استفاده شده است که ویژگی FPN را نیز پشتیبانی می‌کند و تنها از خروجی لایه آخر استفاده نشده و از خروجی سه لایه آخر آن استفاده می‌شود. برای قسمت بعدی یا همان AnchorGenerator از کلاس آماده torchvision استفاده شده است و برای قسمت سائزهای مربوط به هر خروجی backbone، ۴ سائز مختلف با توجه به اندازه stride آن‌ها و برای هر کدام از ۵ نسبت تصویر استفاده شده است که در قسمت زیر نمایش داده شده است.

```
self.anchorGenerator = AnchorGenerator(  
    sizes=((4, 8, 16, 32), (16, 32, 64, 128), (32, 64, 128, 256)),  
    aspect_ratios=((0.2, 0.5, 1.0, 2.0, 5.0),) * 3,  
)
```

در تصویر بالا مشاهده می‌شود که ۴ سائز کوچک از ۴ پیکسل تا ۳۲ پیکسل برای خروجی لایه دوم، ۴ سائز ۱۶ تا ۱۲۸ پیکسل برای خروجی سوم و ۳۲ تا ۲۵۶ پیکسل برای خروجی لایه آخر backbone در نظر گرفته شده است.

سپس برای قسمت region proposal network از کلاس آمده کتابخانه torchvision استفاده شده است که تنظیمات مربوط به این قسمت در تصویر زیر مشاهده می‌شود.

```
self.rpn = RegionProposalNetwork(
    anchor_generator=self.anchorGenerator,
    head=RPNHead(
        in_channels=256,
        num_anchors=self.anchorGenerator.num_anchors_per_location()[0],
    ),
    fg_iou_thresh=0.7,
    bg_iou_thresh=0.3,
    batch_size_per_image=256,
    positive_fraction=0.5,
    pre_nms_top_n={"training": 2000, "testing": 1000},
    post_nms_top_n={"training": 2000, "testing": 1000},
    nms_thresh=0.5,
)
```

در نهایت به قسمت RoIHeads می‌رسیم که برای این بخش نیز از کلاس آمده کتابخانه torchvision استفاده شده است که تنظیمات مربوط به این بخش در تصویر زیر مشاهده می‌شود.

```

self.roi_heads = RoIHeads(
    box_roi_pool=MultiScaleRoIAlign(
        featmap_names=[
            "0",
            "1",
            "2",
        ],
        output_size=14,
        sampling_ratio=2,
    ),
    box_head=TwoMLPHead(in_channels=256 * 14 * 14, representation_size=1024),
    box_predictor=FastRCNNPredictor(in_channels=1024, num_classes=num_classes),
    fg_iou_thresh=0.5,
    bg_iou_thresh=0.5,
    batch_size_per_image=512,
    positive_fraction=0.25,
    score_thresh=0.3,
    nms_thresh=0.3,
    detections_per_img=50,
    bbox_reg_weights=(10.0, 10.0, 5.0, 5.0),
)

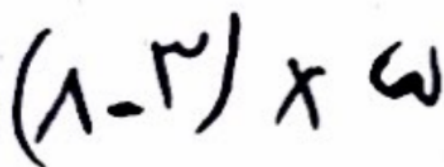
```

همچنین در این بخش برای قسمت‌های `box_head` و `box_predictor` از کلاس‌های آماده استفاده شده است. در ادامه مدل یک متود `forward` دارد که برای آموزش و پیش‌بینی استفاده می‌شود. در این متود ابتدا عکس موردنظر به شبکه `backbone` داده می‌شود سپس خروجی `backbone` به `anchor generator` داده می‌شود. در ادامه خروجی قسمت قبل به بخش `region proposal` داده می‌شود تا کادرهای موردنظر را پیشنهاد داده و درنهایت این کادر یا فیلتر شده و بهترین آن‌ها بازگردانده می‌شوند. اگر این متود در حالت آموزش استفاده شود با استفاده از داده‌های آموزشی، میزان `loss` را برای آپدیت کردن وزن‌ها محاسبه می‌کند ولی اگر در حالت پیش‌بینی اجرا شود، تنها کادرهای محاطی را برمی‌گرداند.

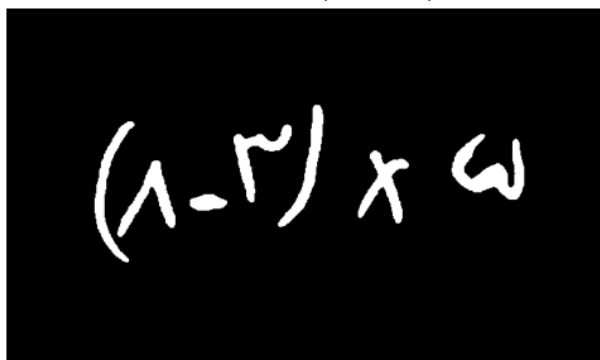
## ساختار کد

برای پیاده‌سازی این بخش، ابتدا نیاز بود که داده‌ها را خوانده و آن‌ها را با توجه به نیاز پردازش کنیم. در ابتدا یک کلاس مربوط به خواندن داده‌ها نوشته شده است که با دریافت آدرس عکس‌ها و فایل‌های `JSON` آن‌ها را در صورت وجود خوانده و برای مراحل بعد پردازش می‌کند. در این بخش با توجه به متغیر بودن سایز عکس‌ها و یکسان نبودن آن‌ها، تعدادی کلاس برای تغییر عکس‌ها و باکس‌ها نوشته شده است که درنهایت امر یک عکس خام را به صورت زیر به عکسی تمیزتر و با سایز مشخصی تبدیل می‌کند.

Original Image



Transformed (Binarized)



در تصویر بالا عکس سمت چپ عکس خام می‌باشد و عکس سمت راست عکس نهایی بعد از اعمال تغییرات می‌باشد. در اثر اعمال این تغییرات، تمامی عکس‌ها به یک‌شکل تبدیل می‌شوند که مدل کار راحتی‌تری داشته باشد.

در ادامه به قسمت backbone می‌رسیم که در قسمت قبل گفته شده است که با الهام از معماری ResNet18 پیاده‌سازی شده است. این معماری در هر لایه از کلاس BasicBlock استفاده می‌کند که شامل دو لایه Convolution، دو لایه Batch Normalization و یک لایه ReLU می‌باشد. در نهایت کلاس Backbone ابتدا داده ورودی را به یک لایه Convolution داده و سپس خروجی آن را به لایه Batch Normalization می‌دهد و در نهایت تابع Relu روی آن اجرا می‌شود. سپس خروجی این قسمت به یک لایه max pooling داده شده و خروجی این بخش نیز به ترتیب به لایه‌های بعدی داده می‌شود که در نهایت سه خروجی لایه آخر Backbone در قسمت‌های بعدی استفاده می‌شود. به دلیل آنکه اندازه خروجی هر لایه متفاوت بوده و از ۱۲۸ تا ۵۱۲ متغیر می‌باشد، تمامی آن‌ها در قسمت FPN به ۲۵۶ تبدیل می‌شوند.

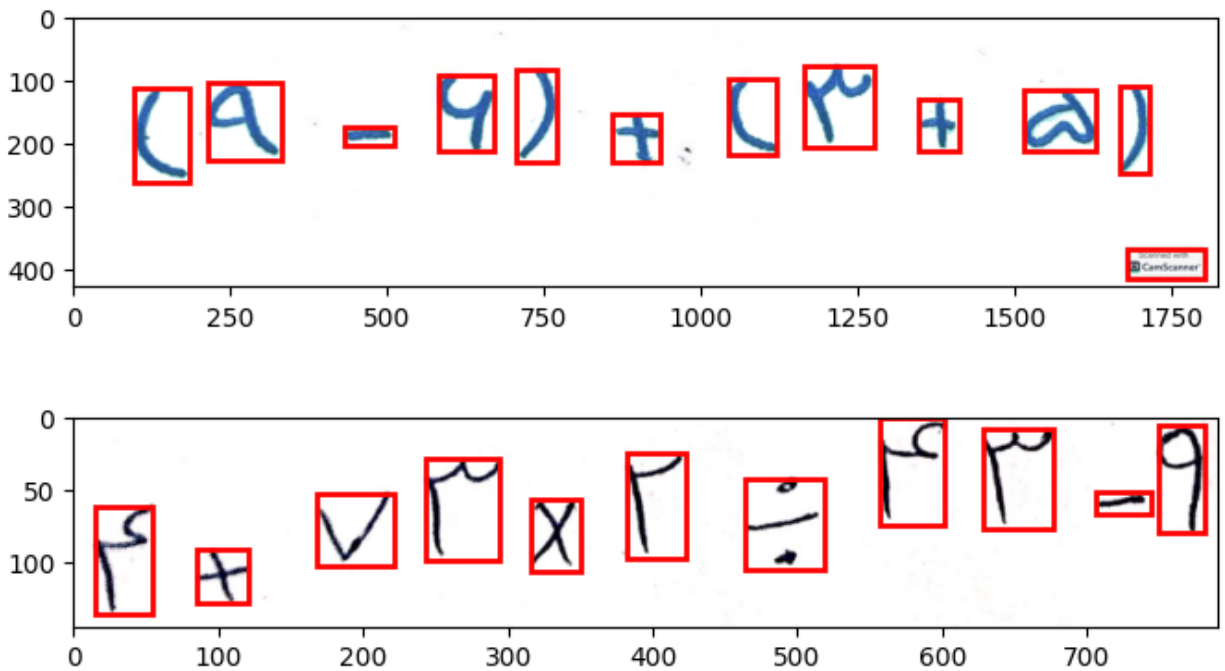
در قسمت قبل در مورد جزئیات مدل بحث شده است که از تکرار آن پرهیز می‌کنیم. در قسمت بعدی به توابع مربوط به train و evaluation می‌رسیم. در این بخش به‌طور خلاصه مدل در هر epoch ابتدا آموزش داده می‌شود و مقدار loss به دست می‌آید سپس با استفاده از داده evaluation، میزان معیارهای IoU و همچنین MaP محاسبه می‌شود. در هر محله اگر نتیجه معیار MaP به دست آمده از مراحل قبل بالاتر باشد، مدل در آن مرحله ذخیره می‌شود و مرحله بعد آغاز می‌شود.

در نهایت با تمام شدن بخش آموزش و ذخیره شدن بهترین مدل، مدل ذخیره شده با داده تست بررسی شده و نتایج به دست آمده در یک فایل CSV ذخیره می‌گردد تا در quera مورد بررسی قرار گیرد.

## آزمون و خطا و تعیین پارامترها

در مسیر انجام این قسمت، ابتدا با استفاده از مدل آماده `faster_rcnn_r_50_fpn`، یک نمونه کد تستی نوشته شد تا با نحوه کار این مدل‌ها و همچنین آماده‌سازی داده برای آموزش و تست، آشنا شویم. سپس با استفاده از این مدل و داده‌های به‌دست‌آمده در قسمت‌های قبل پروژه، مدل را طراحی کردیم. در این بین پارامترهای زیادی باید به مدل داده می‌شود تا میزان کارایی آن افزایش یابد. ابتدا برای درک مفهوم و اثر هر پارامتر، درباره آن‌ها تحقیق کردیم و سپس با شروع از مقادیری که به‌صورت کلی عددهای معقولی هستند مدل را آموزش دادیم و سپس با رسم کردن کادرها و عکس‌ها و با دیدن نواقص و اشتباهات مدل، پارامترها را تغییر دادیم تا به نتایج بهتری دست پیدا کنیم. به‌طور مثال در مورد سایز `anchor` ها، در ابتدا سایزها بزرگ بودند و مدل توانایی تشخیص کاراکترهای کوچک را نداشت به همین دلیل برای بهبود این بخش به استفاده از FPN روی آوردیم که با استفاده از خروجی‌هایی از Backbone که `stride` پایین‌تری دارند و می‌توان سایز `anchor` های کوچک‌تری برای آن‌ها استفاده کرد، بتوانیم کاراکترهای کوچک را تشخیص دهیم. در این بین مصورسازی نتایج و عکس‌ها بسیار کمک‌کننده بودند. برخی از عکس‌ها محو بودند در برخی دیگر خطوط اضافی وجود داشت. همچنین در بعضی از آن‌ها به دلیل اسکن نشدن، سایه در تصویر بود که با استفاده از این مشاهدات توانستیم با نوشتن کلاس‌های `transform`، داده را تمیز کرده و برای پردازش بهینه‌تر کنیم. این مشاهدات و تغییرات توانست میزان `F1-score` را از ۴۹ تا حدود ۷۰ برساند. در ادامه برای بهتر شدن نتایج، پارامترهای نظیر `nms_thresh` که تعیین‌کننده میزان هم‌پوشانی مجاز بین دو باکس است و همچنین `score_thresh` که کمینه مجاز برای مقدار `score` باکس‌های پیشنهادی را تعیین می‌کند، موردبررسی قرار گرفتند و مقدار ۰.۳ برای هر دو انتخاب‌شده است. در نهایت با این تغییرات میزان `F1-score` تا ۷۵ درصد بالا رفت ولی پس از آن با هر تغییری در پارامترها، این عدد یا کمتر می‌شد یا تغییری نمی‌کرد. برای آنکه بتوانیم مدل را بهبود ببخشیم به افزایش تعداد داده‌های آموزش روی آوردیم. تصمیم گرفتیم تا هر عکس را با ۶ سایز مختلف در بخش آموزش به مدل بدهیم و ۳۰۰ عکس بخش آموزش را با این کار به ۱۸۰۰ عکس تبدیل کنیم. این کار در کلاس `ExpressionDataset` که وظیفه خواندن و فراهم کردن داده را دارد انجام‌شده است به این صورت که اگر این کلاس برای خواندن داده آموزش استفاده شود، با استفاده از این شش `[512, 640, 768, 896, 1024, 1500]` سایز مختلف، از هر تصویر ۶ تصویر ایجاد می‌کند که مقدار عرض آن عکس‌ها برابر این اعداد می‌باشد و مقدار ارتفاع با حفظ نسبت تصویر اولیه تغییر می‌کند. این امر باعث افزایش معیار `F1-score` از ۷۵ به ۷۸ شد ولی زمان لازم برای آموزش را افزایش داد که با توجه به میزان پیشرفت حاصله، کاملاً قابل توجیه می‌باشد.

در قسمت زیر دو نمونه از تصاویر بخش تست با کادرهای پیش‌بینی‌شده توسط مدل مشاهده می‌شود.



## معیار IoU

این معیار بررسی می‌کند که کادرهای پیشنهادی به چه میزان با کادرهای اصلی همپوشانی دارد. در سایت Quera ۵ آستانه مختلف در نظر گرفته شده است که میزان IoU با آن آستانه‌ها مقایسه می‌شود. نتایج به‌دست‌آمده به‌صورت زیر است.

```
test 1
97.17 out of 100
F1-score for IoU_threshold = 0.5

test 2
94.61 out of 100
F1-score for IoU_threshold = 0.6

test 3
88.94 out of 100
F1-score for IoU_threshold = 0.7

test 4
75.49 out of 100
F1-score for IoU_threshold = 0.8

test 5
35.8 out of 100
F1-score for IoU_threshold = 0.9
```

در تصویر بالا مشاهده می‌شود که ۹۷ درصد کادرها حداقل ۵۰ درصد با کادرهای اصلی همپوشانی داشته و در نهایت ۳۵.۸ درصد کادرهای پیشنهادی حداقل ۹۰ درصد همپوشانی دارند.

## بخش دوم

### بخش دوم - یک - روش‌های استخراج ویژگی

داده‌های مورد استفاده در این پروژه شامل تصاویر دست‌نویس اعداد فارسی، پرانتزها و نمادهای چهار عمل اصلی ریاضی (+، -، ×، ÷) هستند. این نوع داده‌ها معمولاً با چالش‌هایی همچون تفاوت در ضخامت قلم، تغییرات شکل حروف و اعداد بین نویسندگان مختلف، نویز ناشی از اسکن یا تصویربرداری و اتصال یا همپوشانی کاراکترها همراه هستند.

به همین دلیل، پیش از هرگونه پردازش، لازم است تصاویری با کیفیت و شکل استاندارد از هر کاراکتر تهیه شود تا مدل‌های پردازش تصویر بتوانند بر اساس ویژگی‌های بصری پایدار و قابل‌اعتماد آموزش ببینند.

### مرحله پیش‌پردازش

فرآیند پیش‌پردازش در این پروژه شامل چند مرحله کلیدی بوده است:

۱. برش کاراکترها از تصویر اصلی با استفاده از مختصات کادرهای محاطی (Bounding Box) موجود در برچسب‌ها.

۲. حذف خطوط افقی ناخواسته که ممکن است ناشی از ساختار جدول‌ها یا خط‌کش‌ها باشد.

۳. پاک‌سازی نویزهای متصل به لبه‌ها و اجزای کوچک غیرمرتبط از تصویر کاراکتر.

۴. باینری‌سازی (سیاه‌وسفید کردن) تصویر با استفاده از آستانه‌گذاری اتسو (Otsu Thresholding) برای برجسته‌کردن کاراکتر.

۵. نرمال‌سازی ابعاد تصویر به سایز ثابت ۲۸×۲۸ پیکسل و اعمال Anti-aliasing برای حفظ کیفیت لبه‌ها.

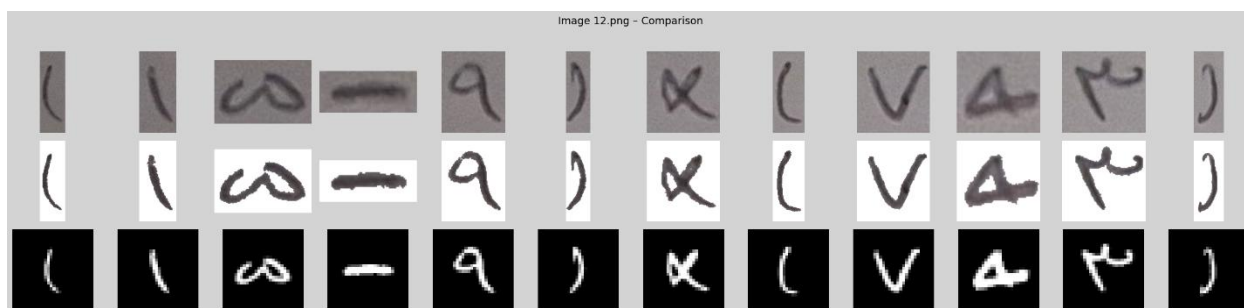
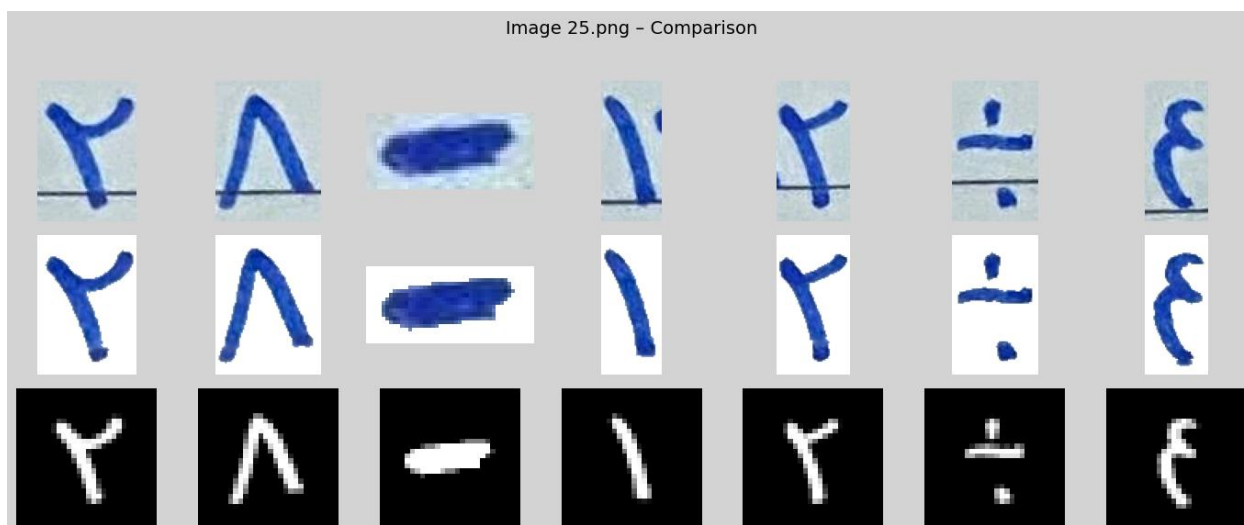
این مراحل باعث شدند که ورودی استخراج ویژگی‌ها یک تصویر استاندارد و هم‌ابعاد باشد که اثر تغییرات نویسنده و کیفیت اسکن را به حداقل برساند.



$$20 + (9 \div (7 - 3))$$

$$20 + (9 / (7 - 3))$$

این تصویر یک نمونه از داده‌های کادربندی شده (Bounding Box) است که برای مسئله‌ی ما استفاده می‌شود. همان‌طور که در تصویر مشاهده می‌کنید، هر کاراکتر (عدد، عملگر ریاضی، یا پرانتز) با یک کادر قرمز مشخص شده و در پایین تصویر، عبارت کامل به صورت متنی نمایش داده شده است که مطابق تصاویر زیر تمیز شده اند:



## روش‌های استخراج ویژگی به‌کاررفته

### ۱. ویژگی‌های شکلی (Shape-based Features)

در این روش، مجموعه‌ای از ویژگی‌های هندسی و آماری از شکل کلی کاراکتر استخراج می‌شود که بیانگر خصوصیات کلی و ساختاری آن هستند. برخی از این ویژگی‌ها عبارت‌اند از:

- مساحت (Area) و محیط (Perimeter) کاراکتر
  - ابعاد کادر محاطی (Bounding Box Width/Height) و نسبت طول به عرض (Aspect Ratio)
  - چگالی لبه‌ها (Edge Density) و تعداد پیکسل‌های لبه (Edge Count) با استفاده از الگوریتم Canny
  - گردی (Circularity) برای سنجش میزان دایره‌ای بودن شکل
  - ممان‌های هفت‌گانه Hu به‌عنوان توصیف‌گرهای نامتغیر نسبت به چرخش، انتقال و مقیاس
  - نسبت جرم عمودی و افقی برای مقایسه توزیع پیکسل‌ها در نیمه‌های مختلف تصویر
- این ویژگی‌ها کمک می‌کنند که مدل تفاوت بین اشکال هندسی ساده (مثل ۰ و O) یا بین پرانتز و سایر نمادها را تشخیص دهد.

### ۲. الگوهای باینری محلی (Local Binary Patterns - LBP)

روش LBP یک توصیف‌گر بافت (Texture Descriptor) است که الگوهای محلی پیکسل‌ها را بر اساس مقایسه شدت روشنایی پیکسل مرکزی با همسایگانش ثبت می‌کند.

در این پروژه، از LBP با شعاع ۱ و تعداد نقاط همسایه ۸ استفاده شده است. پس از محاسبه مقادیر LBP برای هر پیکسل، هیستوگرام توزیع این مقادیر ساخته می‌شود و به‌عنوان ویژگی به مدل داده می‌شود.

این روش برای تمایز کاراکترهایی که ساختار بافتی متفاوت دارند (مثلاً خط صاف علامت منها در مقابل منحنی‌های عدد ۳) بسیار مفید است.

### ۳. گرادیان‌های جهت‌دار هیستوگرامی (Histogram of Oriented Gradients - HOG)

HOG یک روش قدرتمند برای استخراج ویژگی بر اساس جهت لبه‌ها و توزیع گرادیان‌ها در تصویر است. در این پروژه، تصویر به سلول‌هایی با اندازه  $8 \times 8$  پیکسل تقسیم شده و برای هر سلول، هیستوگرام جهت گرادیان محاسبه می‌شود. سپس سلول‌ها در بلوک‌های  $2 \times 2$  نرمال‌سازی می‌شوند تا اثر تغییر روشنایی کاهش یابد. HOG به‌ویژه برای تمایز بین کاراکترهایی با ساختار لبه متفاوت (مثل  $\times$  در مقابل  $+$ ) بسیار کارآمد است، زیرا الگوی جهت لبه‌ها در این کاراکترها متفاوت و قابل تفکیک است.

○ برای اجرای کد این بخش در فولدر src دستور زیر را اجرا کنید:

```
python -m part2 --features shape,lbp,hog --hog-pca 48 --k-min 12 --k-max 24
```

## بخش دوم - دو - روش‌های خوشه‌بندی به کار رفته و پارامترهای آن

پس از استخراج ویژگی‌های شکلی، بافتی و گرادیانی از تصاویر کاراکترها، لازم است این بردارهای ویژگی به گروه‌های مشابه تقسیم شوند تا بتوان نمونه‌های همسان را در یک خوشه قرار داد. این فرآیند بدون استفاده از برچسب (Label) انجام می‌شود و هدف آن شناسایی ساختار درونی داده‌ها بر اساس شباهت ویژگی‌ها است.

در این پروژه، از الگوریتم K-Means به عنوان روش اصلی خوشه‌بندی استفاده شده و به منظور انتخاب تعداد خوشه بهینه، از تحلیل‌های Elbow و Silhouette Score کمک گرفته شده است.

### روش خوشه‌بندی به کار رفته

#### ۱. الگوریتم K-Means

K-Means یکی از رایج‌ترین الگوریتم‌های خوشه‌بندی است که بر اساس کمینه‌کردن مجموع فاصله نقاط تا مراکز خوشه (Centroids) عمل می‌کند. روند کلی آن به صورت زیر است:

۱. انتخاب K مرکز اولیه (به صورت تصادفی یا با روش‌های بهینه‌تر مانند ++k-means)

۲. انتساب هر نمونه به نزدیک‌ترین مرکز خوشه بر اساس فاصله اقلیدسی

۳. به‌روزرسانی مراکز خوشه به عنوان میانگین نقاط موجود در هر خوشه

۴. تکرار مراحل ۲ و ۳ تا همگرایی (تغییر نکردن مراکز یا رسیدن به حداکثر تعداد تکرار)

در این پروژه، قبل از اجرای K-Means تمام ویژگی‌ها به‌طور جداگانه استانداردسازی (StandardScaler) شدند. ویژگی‌های HOG پس از استانداردسازی با PCA به ۴۸ مؤلفه کاهش شدند تا ابعاد بالا باعث نویز و کاهش دقت خوشه‌بندی نشود و در نهایت، تمام ویژگی‌ها با L2 Normalization نرمال‌سازی شدند.

### انتخاب تعداد خوشه (K)

برای تعیین مقدار بهینه K، بازه‌ای بین ۱۲ تا ۲۴ بررسی شد. دو معیار اصلی انتخاب:

- روش Elbow: رسم نمودار اینرسی (Inertia) بر حسب K و یافتن نقطه زانویی که کاهش اینرسی پس از آن ناچیز می‌شود.
- امتیاز Silhouette: سنجش میزان جداسازی خوشه‌ها؛ هرچه این مقدار به ۱ نزدیک‌تر باشد، مرزبندی خوشه‌ها بهتر است.

بهترین K با توجه به بالاترین مقدار Silhouette و تأیید بصری نمودار Elbow انتخاب شد.

### پارامترهای به‌کاررفته در K-Means

n\_clusters: مقدار انتخاب شده K (به‌دست آمده از تحلیل Elbow/Silhouette)

n\_init = 20: اجرای الگوریتم با ۲۰ مقدار اولیه تصادفی برای کاهش احتمال گیر افتادن در کمینه محلی

random\_state = 42: برای بازتولید نتایج

max\_iter: مقدار پیش‌فرض کتابخانه Scikit-learn (۳۰۰ تکرار)

✓ استفاده از PCA برای کاهش بعد و حذف نویز از ویژگی‌های HOG باعث بهبود جداسازی خوشه‌ها شد.

✓ ترکیب ویژگی‌های شکلی، بافتی و گرادیانی باعث شد خوشه‌بندی نه تنها بر اساس هندسه، بلکه بر اساس الگوهای محلی و جهت لبه‌ها نیز انجام شود.

✓ بررسی همزمان Elbow و Silhouette از انتخاب K نامناسب جلوگیری کرد.

## بخش دوم – سه – نمایش اعضای نزدیک به مرکز و نمونه‌های دورافتاده خوشه‌ها

برای ارزیابی کیفی خوشه‌بندی، تنها تکیه بر شاخص‌های عددی مانند Silhouette یا Davies-Bouldin کافی نیست. مشاهده بصری نمونه‌های واقعی که در هر خوشه قرار گرفته‌اند، به درک بهتر عملکرد الگوریتم کمک می‌کند.

در این پروژه، برای هر خوشه، نمونه‌های نزدیک‌ترین به مرکز خوشه و نمونه‌های دورافتاده شناسایی و نمایش داده شدند.

### معیار نزدیکی و دوری از مرکز خوشه

پس از اجرای K-Means، مرکز هر خوشه (Centroid) در فضای ویژگی‌ها محاسبه شد.

برای هر تصویر در آن خوشه:

۱. فاصله اقلیدسی (Euclidean Distance) بین بردار ویژگی آن تصویر و مرکز خوشه محاسبه شد.

۲. نمونه‌ها بر اساس این فاصله مرتب شدند.

۳. ۵ نمونه با کمترین فاصله به عنوان نزدیک‌ترین به مرکز انتخاب شدند.

۴. ۵ نمونه با بیشترین فاصله به عنوان دورافتاده‌ترین اعضای خوشه شناسایی شدند.

### اهمیت این نمایش

اعضای نزدیک به مرکز خوشه نماینده بهترین نمونه‌های آن دسته هستند و نشان می‌دهند که ویژگی‌های استخراج‌شده چقدر توانسته‌اند الگوی اصلی آن دسته را ثبت کنند.

اعضای دورافتاده (Outliers) معمولاً شامل نویز، کاراکترهای ناخوانا یا نمونه‌هایی هستند که ویژگی‌هایشان با هیچ خوشه‌ای به‌طور کامل همخوانی ندارد. بررسی این نمونه‌ها می‌تواند به بهبود پیش‌پردازش یا انتخاب ویژگی کمک کند.

## نمایش و خروجی‌ها

برای هر خوشه، تصاویری از نزدیک‌ترین و دورافتاده‌ترین اعضا به‌صورت جداگانه ذخیره و بررسی شد:

- پوشه `results/part2/clusters` شامل پوشه‌هایی با نام `cluster_id` است که همه اعضای هر خوشه را نگهداری می‌کند.
- نمونه‌ها در این پوشه‌ها بر اساس برچسب خوشه دسته‌بندی شدند و امکان مرور مستقیم آنها فراهم است.
- با مقایسه بصری این نمونه‌ها، می‌توان الگوهای غالب هر خوشه و موارد استثنا را تشخیص داد.

○ با اجرای کد زیر در فولدر `src` نتایج نزدیک و دور به مرکز دسته معلوم شد:

```
python -m part2.samples_viz
```

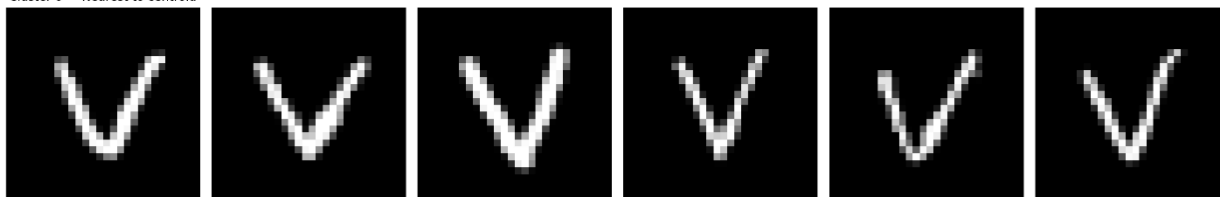
9

```
python -m part2.samples_viz_grid
```

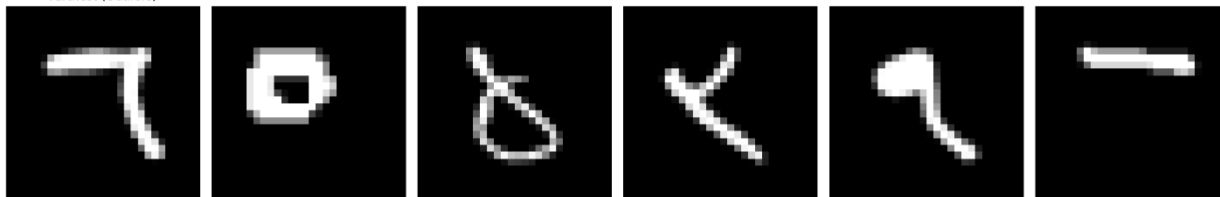


نتایج حاصل از این بخش:

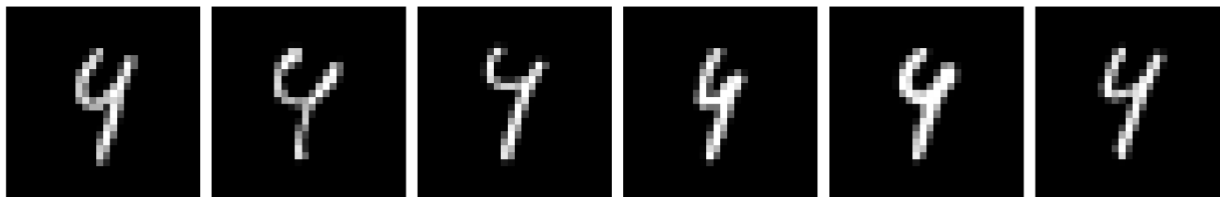
Cluster 0 — Nearest to centroid



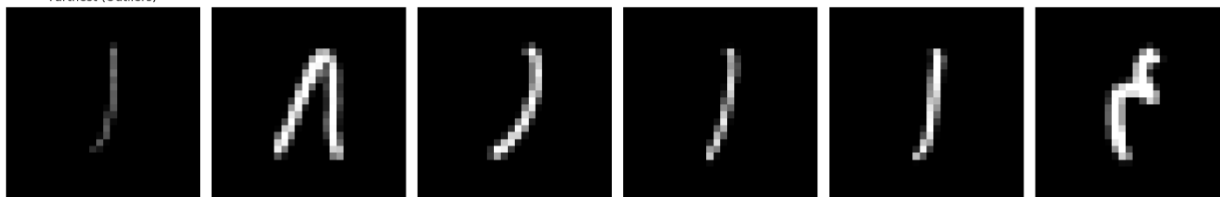
Farthest (Outliers)



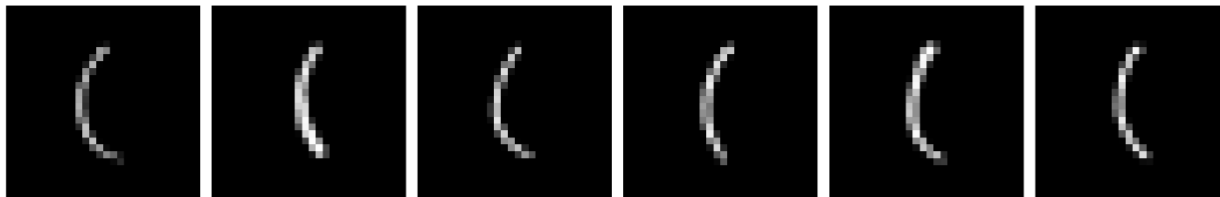
Cluster 1 — Nearest to centroid



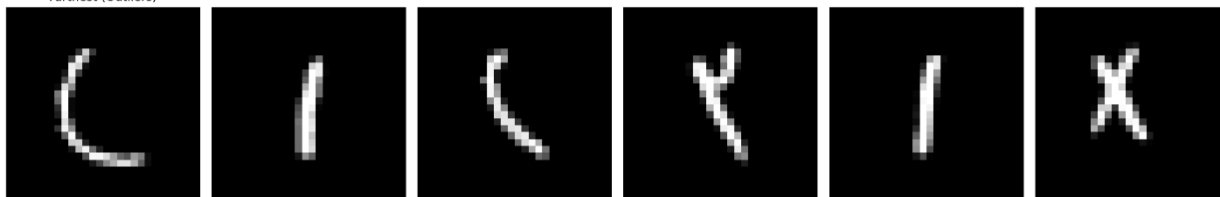
Farthest (Outliers)



Cluster 2 — Nearest to centroid



Farthest (Outliers)



▪ باقی موارد درون فولدر `results\part2\plots` موجود هستند.

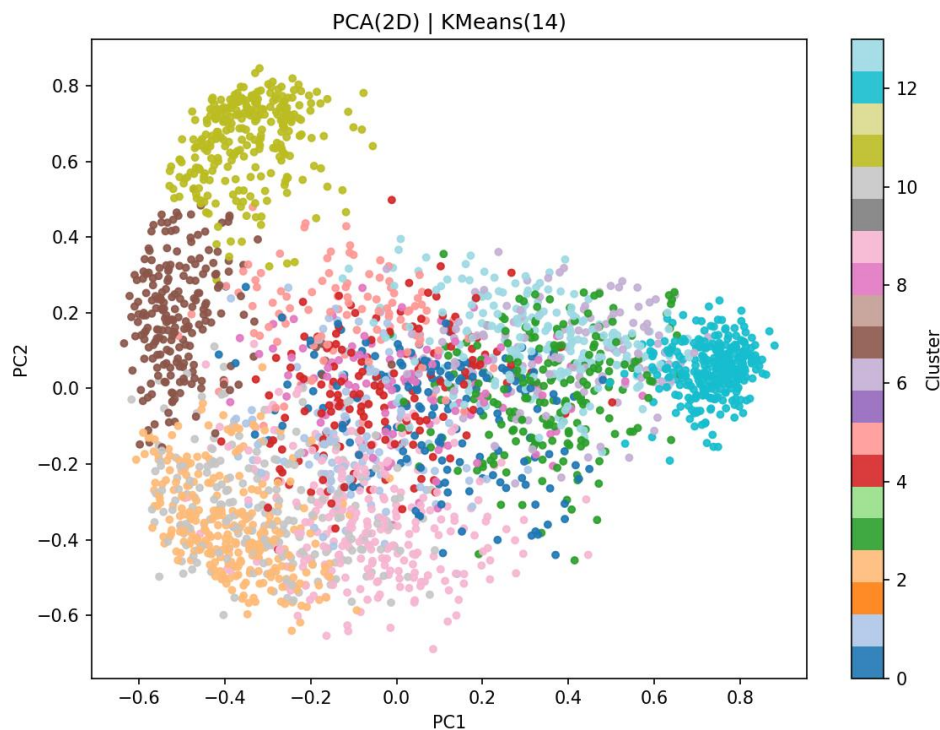
## بحث درباره کیفیت نتایج خوشه‌بندی

### ۱. ارزیابی ظاهری

با بررسی تصاویر نمونه‌ی نزدیک‌ترین اعضا به مرکز هر خوشه و همچنین دورافتاده‌ترین اعضا (Outliers)، مشخص شد که اکثر خوشه‌ها از نظر شباهت ظاهری بین اعضای مرکزی، همگن هستند. به‌طور کلی، نمونه‌های نزدیک به مرکز خوشه وضوح بالایی از الگوهای مشترک در ویژگی‌ها (شکل، LBP، و HOG کاهش‌بُعد یافته) را نشان می‌دهند.

در مقابل، نمونه‌های دورافتاده معمولاً حاوی نویز، تغییرات شدید در اندازه یا نسبت تصویر، و یا دست‌خط‌های غیرمعمول بوده‌اند که باعث فاصله زیاد آن‌ها از مرکز خوشه شده است.

پراکندگی داده‌ها در فضای دوبعدی کاهش‌بُعد یافته (شکل ۱) نشان می‌دهد که خوشه‌ها تا حدی از هم تفکیک‌پذیر هستند، ولی در برخی نواحی همپوشانی نسبی بین خوشه‌ها مشاهده می‌شود. این همپوشانی عمدتاً در نمونه‌هایی دیده می‌شود که ویژگی‌های مرزی بین دو یا چند کلاس دارند.



شکل ۱ - پراکندگی خوشه‌ها در فضای دوبعدی (PCA Scatter Plot)

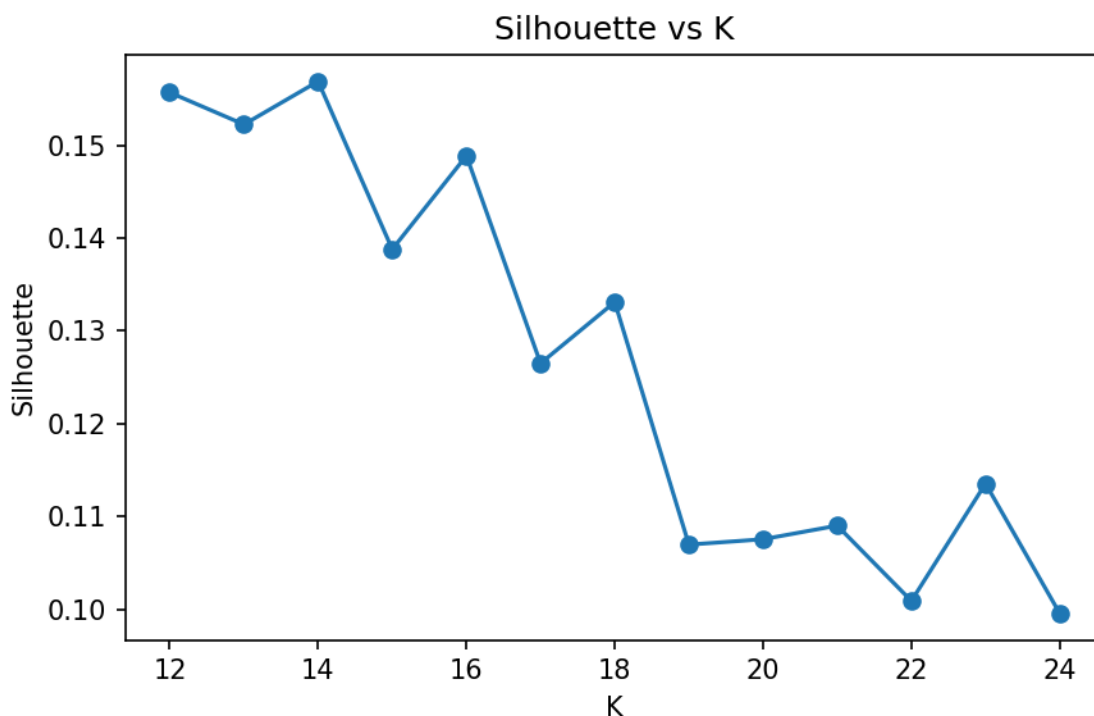
## ۲. ارزیابی کمی

برای ارزیابی عددی، مقادیر شاخص‌ها به صورت زیر به دست آمدند:

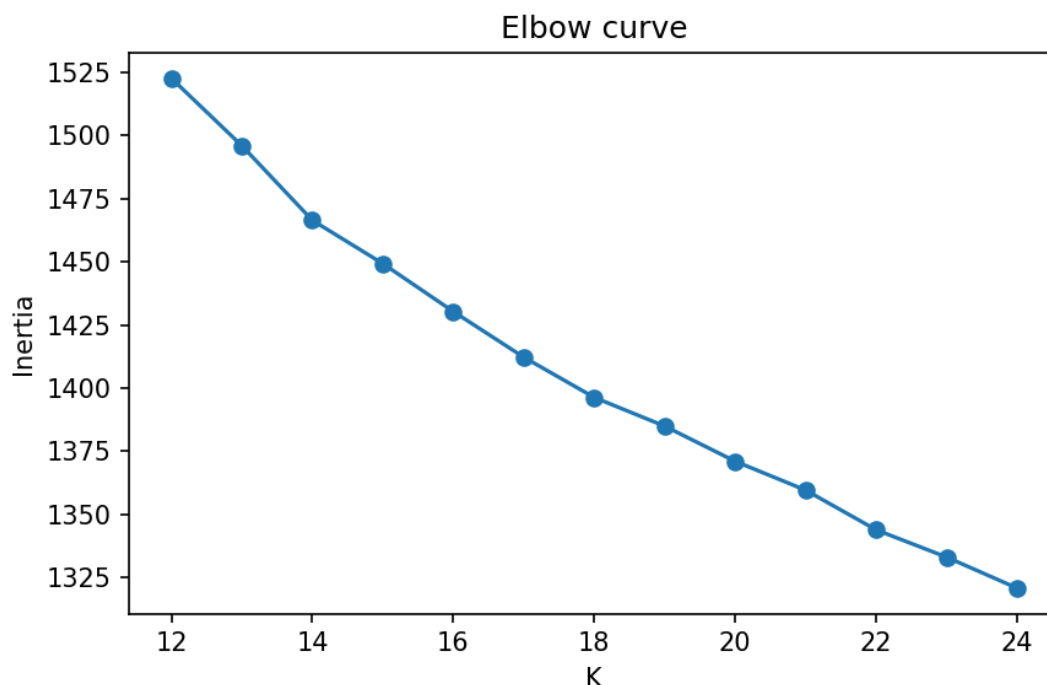
- Silhouette Score: مقدار ۰.۱۵۶۹ برای  $K=14$ ، که نشان‌دهنده جدایی متوسط بین خوشه‌ها و همگنی درون خوشه‌ها است.
- Davies–Bouldin Index: مقدار ۲.۲۳۵۸، که مقادیر پایین‌تر نشان‌دهنده خوشه‌های متمایزتر است.
- Calinski–Harabasz Index: مقدار ۱۸۱.۵۶، که مقدار بالاتر آن عموماً بیانگر خوشه‌بندی بهتر است.

دو روش انتخاب  $K$  به کار گرفته شد:

- روش Silhouette (شکل ۲) که مقدار  $K=14$  را به عنوان بهترین گزینه پیشنهاد داد.
- روش Elbow (شکل ۳) که نقطه شکست منحنی اینرسی را در  $K=22$  تعیین کرد.



شکل ۲ - تغییرات امتیاز سیلوئت بر اساس تعداد خوشه‌ها



شکل ۳ - روش Elbow برای انتخاب تعداد خوشه

با توجه به تعادل بین تفکیک پذیری و اندازه خوشه‌ها،  $K=14$  به عنوان مقدار نهایی انتخاب شد.

### جمع بندی

نتایج خوشه بندی با  $K=14$  در این پروژه، با وجود پیچیدگی و نویز ذاتی داده‌ها، توانسته است تا حد قابل قبولی گروه‌های معنادار ایجاد کند. تحلیل دیداری تأیید می‌کند که نمونه‌های مرکزی هر خوشه از نظر بصری همگن هستند، هرچند وجود برخی خوشه‌های همپوشان و نمونه‌های مرزی طبیعی است. معیارهای عددی نیز این ارزیابی را پشتیبانی می‌کنند و نشان می‌دهند که مدل انتخاب شده توازن مناسبی بین تعداد خوشه‌ها و کیفیت جداسازی ایجاد کرده است.

## بخش سوم

این گزارش، پیاده سازی جامع روش های یادگیری نیمه نظارتی برای شناسایی کاراکتر های عبارات ریاضی را ارائه می دهد. با چالش کمبود داده های برچسب خورده (تنها ۸ تصویر برچسب دار از میان ۳۰۰ نمونه آموزشی)، ما یک رویکرد self training را با استفاده از معماری اصلاح شده ResNet18 توسعه دادیم. روش ما به دقت 92.3% در سطح کاراکتر و ۸۶.۸% شباهت در سطح عبارت روی مجموعه اعتبارسنجی دست یافت که بهبود قابل توجهی نسبت به خطوط پایه صرفاً نظارتی نشان می دهد.

این پیاده سازی با ترکیب روش شبه برچسب گذاری مبتنی بر اطمینان، تقویت داده تهاجمی و پردازش پس مرحله ای مبتنی بر قواعد، یک راهکار انتها به انتها و مقاوم برای شناسایی عبارات ریاضی ایجاد کرده است.

مشارکت های کلیدی شامل: (۱) یک استراتژی مؤثر شبه برچسب گذاری با دو آستانه، (۲) یک خط لوله پیش پردازش داده جامع بهینه شده برای نماد های ریاضی، و (۳) اعتبارسنجی تجربی مزایای یادگیری نیمه نظارتی در سناریو های فوق العاده کم منبع.

### ۱. مقدمه :

#### ۱.۱. بیان مسئله:

شناسایی عبارات ریاضی، نقطه ی تلاقی چالش بر انگیز بین بینایی ماشین و محاسبات نمادین است که نیازمند مدل هایی است که هم بتوانند کاراکتر های منفرد را با دقت تشخیص دهند و هم انسجام ساختاری کل عبارت را حفظ کنند. این پروژه به طور خاص چالش شناسایی عبارات ریاضی دست نویس را در شرایط کمبود شدید داده های برچسب خورده مورد بررسی قرار می دهد.

این وظیفه چند چالش منحصر به فرد دارد:

**کمبود شدید برچسب ها :** تنها ۸ تصویر از میان 300 تصویر آموزشی دارای برچسب ground truth برای عبارت ها هستند.

**تنوع کاراکترها :** 16 کلاس کاراکتر متفاوت شامل ارقام ۰ تا ۹ و عملگر های ( ) و + و - و \* و /

پیچیدگی ساختاری : عبارات ریاضی نیازمند رعایت اعتبار نحوی فراتر از دقت در سطح کاراکتر هستند.

ابهام دیداری : کاراکتر های مشابه از نظر ظاهری (مثل «۶» در مقابل «۹»، یا «)» در مقابل «(») نیازمند تمایز دقیق و جزئی هستند.

## ۱.۲. مرور کلی مجموعه داده:

ساختار این مجموعه داده بازتابی از سناریو های واقعی است که در آن به دست آوردن داده های برچسب خورده هزینه بر است.

Table 1: Dataset Distribution and Labeling Status

Split	Image Range	Total Images	Bounding Boxes	Expression Labels
Training (Labeled)	0-7	8	✓	✓
Training (Unlabeled)	8-299	292	✓	×
Validation	300-499	200	✓	✓
Test	500-699	200	✓	×

## ۱.۳. مرور کلی رویکرد :

راهکار ما از یک پایپلاین یادگیری نیمه نظارتی مبتنی بر self-training استفاده می کند که بهره وری هر دو نوع داده برچسب خورده و بدون برچسب را به حداکثر می رساند.

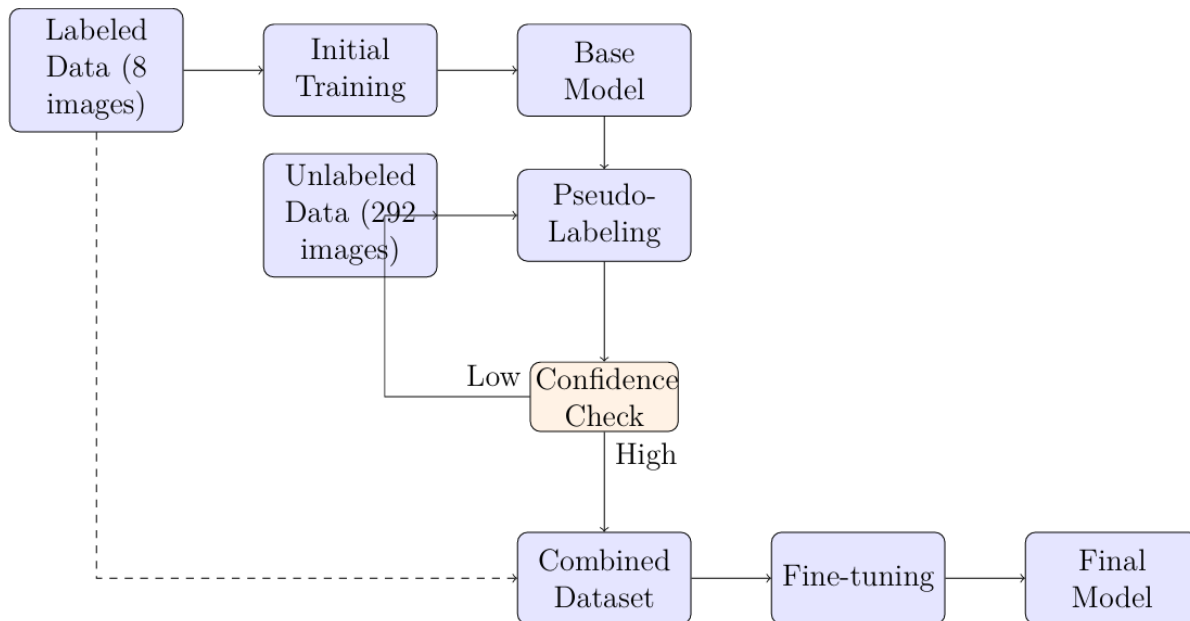


Figure 1: Semi-Supervised Learning Pipeline Overview

#### ۱.۴. مشارکت‌ها

این کار دارای مشارکت‌های زیر است:

شبه برچسب گذاری با دو آستانه : یک استراتژی نوین انتخاب مبتنی بر اطمینان که هم اطمینان مطلق و هم جداسازی بین کلاسی را در نظر می‌گیرد.

پایپلاین پیش پردازش جامع : تکنیک های پردازش تصویر تخصصی بهینه شده برای شناسایی کاراکترهای ریاضی

اعتبارسنجی تجربی : آزمایش های گسترده که نشان دهنده ی بهبود ۴.۸٪ در نرخ تطابق کامل از طریق یادگیری نیمه نظارتی است.

پیاده سازی متن باز : کدی ماژولار و مستند که برای باز تولید و گسترش مناسب است.

## ۲. یادگیری نیمه نظارتی و مدل

### ۲.۱. مبانی نظری

#### ۲.۱.۱. چارچوب خود-یادگیری یا self-training

خود-یادگیری، که با نام‌های خود-برچسب گذاری یا شبه برچسب گذاری نیز شناخته می‌شود، یک روش پوششی است که به صورت تکراری از پیش‌بینی های خود مدل برای گسترش مجموعه آموزشی استفاده می‌کند. توجه نظری این رویکرد بر فرض خوشه‌ای تکیه دارد: مرزهای تصمیم باید در نواحی با چگالی پایین فضای ورودی قرار گیرند.

داده های داده شده:

Labeled dataset:  $\mathcal{D}_L = \{(x_i, y_i)\}_{i=1}^{n_L}$  where  $n_L = 8$  images

Unlabeled dataset:  $\mathcal{D}_U = \{x_j\}_{j=1}^{n_U}$  where  $n_U = 292$  images

Model  $f_\theta: \mathcal{X} \rightarrow \mathcal{Y}$  parameterized by  $\theta$

که هدف self-training را میتوان به شکل زیر فرموله بندی کرد:

$$\mathcal{L}_{total} = \mathcal{L}_{sup}(\mathcal{D}_L) + \lambda \cdot \mathcal{L}_{pseudo}(\mathcal{D}_U, \hat{y})$$

که در آن  $\hat{y} = \arg \max_y P(y|x; \theta)$  برای پیش بینی های با اطمینان بالا و  $\lambda$  میزان تاثیر شبه برچسب ها را کنترل می کند.

#### ۲.۱.۲. استراتژی انتخاب مبتنی بر اطمینان :

معیار انتخاب دو آستانه ای با اطمینان بالا و مرزهای تصمیم شفاف را تضمین می‌کند.



$$\text{Accept pseudo-label if: } \begin{cases} P(y^*|x) > \tau_1 & (\text{Absolute confidence}) \\ P(y^*|x) - P(y^{(2)}|x) > \tau_2 & (\text{Margin confidence}) \end{cases}$$

که در اینجا  $y^*$  کلاس پیش بینی شده و  $y^{(2)}$  دومین پیش بینی بالاتر و  $\tau_1 = 0.96$  و  $\tau_2 = 0.35$  هستند.

## ۲.۲. معماری مدل: ResNet18 اصلاح شده

### ۲.۲.۱. تغییرات معماری :

ما ResNet18 را برای دسته بندی کاراکتر با تغییرات زیر تطبیق دادیم:

---

#### Algorithm 1 ResNet18 Architecture Modifications

---

**Input:** Image tensor  $x \in \mathbb{R}^{3 \times 224 \times 224}$

**Backbone:** Standard ResNet18 layers (conv1  $\rightarrow$  layer4)

**Modified Classifier:**

Linear(512, 256)  $\rightarrow$  BatchNorm1d  $\rightarrow$  ReLU  $\rightarrow$  Dropout(0.5)

Linear(256, 128)  $\rightarrow$  BatchNorm1d  $\rightarrow$  ReLU  $\rightarrow$  Dropout(0.3)

Linear(128, 16) // 16 character classes

**Output:** Logits  $z \in \mathbb{R}^{16}$

---

### ۲.۲.۲. منطق معماری :

انتخاب های معماری بر اساس دلایل زیر انجام شده‌اند:

**بدون پیش تمرین:** مقداردهی اولیه تصادفی، هم با قوانین مسابقه سازگار است و هم مزایای خالص یادگیری نیمه نظارتی را نشان می دهد.

**دسته بند چند لایه :** کاهش تدریجی ابعاد (۱۶ $\rightarrow$ ۱۲۸ $\rightarrow$ ۲۵۶ $\rightarrow$ ۵۱۲) باعث انتزاع بهتر ویژگی ها می شود.

نرمال سازی دسته ای یا **Batch Normalization**: آموزش را در شرایط داده محدود پایدار می کند و امکان استفاده از نرخ یادگیری بالاتر را فراهم می سازد.

**Dropout Regularization: Dropout** ۲ مرحله ای ۰.۵ و ۰.۳ از overfitting روی مجموعه لیبل دار کوچک جلوگیری میکند.

### ۲.۳. پایپلاین پیش پردازش داده :

۲.۳.۱. پیش پردازش در سطح کاراکتر: پایپلاین پیش پردازش ما به چالش های خاص در شناسایی کاراکتر های ریاضی میپردازد.

---

#### Algorithm 2 Character Image Preprocessing Pipeline

---

**Require:** Character image  $I_{raw}$

**Ensure:** Preprocessed tensor  $I_{processed} \in \mathbb{R}^{3 \times 224 \times 224}$

$I_{gray} \leftarrow \text{ConvertToGrayscale}(I_{raw})$

$I_{enhanced} \leftarrow \text{CLAHE}(I_{gray}, \text{clipLimit}=2.0, \text{tileSize}=(4,4))$

$I_{binary} \leftarrow \text{AdaptiveThreshold}(I_{enhanced}, \text{method}=\text{OTSU})$

$I_{resized} \leftarrow \text{Resize}(I_{binary}, \text{size}=(224, 224))$

$I_{rgb} \leftarrow \text{ConvertToRGB}(I_{resized})$

$I_{normalized} \leftarrow \text{Normalize}(I_{rgb}, \text{mean}=[0.485, 0.456, 0.406], \text{std}=[0.229, 0.224, 0.225])$

**return**  $I_{processed} \leftarrow \text{Transpose}(I_{normalized}, \text{order}=(2,0,1))$

---

۲.۳.۲. استراتژی افزایش داده: با توجه به کمبود شدید برچسب ها، ما از افزایش داده تهاجمی با احتمال های تنظیم شده دقیق استفاده می کنیم.

Table 2: Data Augmentation Pipeline Configuration

Augmentation	Parameters	Probability	Purpose
Affine Transform	Rotation: $\pm 12$ , Scale: $0.9-1.1 \times$	0.75	Geometric variance
Brightness/Contrast	$\alpha \in [0.8, 1.2]$ , $\beta \in [-0.1, 0.1]$	0.50	Illumination robustness
Gaussian Noise	$\sigma \sim \mathcal{U}(0, 0.05)$	0.40	Noise resilience
Gaussian Blur	Kernel $\in \{3, 5\}$	0.30	Focus variation
Random Erasing	Area: 5-25% of image	0.50	Occlusion handling

## ۲.۴. استراتژی آموزش :

۲.۴.۱. فرآیند آموزش دو مرحله ای: استراتژی آموزشی ما با دقت بین اکتشاف و بهره‌برداری تعادل برقرار می‌کند.

### مرحله اول: آموزش اولیه نظارتی یا Initial Supervised Training:

هدف : یادگیری نمایش های پایه ای کاراکتر ها از داده های برچسب خورده محدود.

- Epochs: 40
- Learning rate:  $1 \times 10^{-3}$  with ReduceLROnPlateau
- Augmentation multiplier:  $30 \times$
- Optimizer: AdamW with weight decay  $1 \times 10^{-4}$

### مرحله دوم: Semi-Supervised Fine-tuning

هدف : افزودن شبه برچسب ها با اطمینان بالا

- Epochs: 20
- Learning rate:  $5 \times 10^{-4}$  (reduced for stability)
- Augmentation multiplier:  $15 \times$  (reduced to prevent overfitting)
- Dataset: Original labeled + pseudo-labeled samples

۲.۴.۲. کنترل کیفیت شبه برچسب‌ها: برای اطمینان از کیفیت شبه برچسب‌ها، ما چندین مکانیزم حفاظتی پیاده‌سازی کرده ایم:

$$\text{Quality Score}(x) = \alpha \cdot P(y^*|x) + \beta \cdot (P(y^*|x) - P(y^{(r)}|x)) + \gamma \cdot H(P(y|x))$$

که در آن  $H$  آنترپی است و مقادیر  $\alpha = 0.5$  و  $\beta = 0.3$  و  $\gamma = -0.2$  در نظر گرفته شده اند.

### ۳. ساختار کد و پیاده‌سازی

#### ۳.۱. اصول طراحی معماری:

الف ( مازولار بودن : اصل مسئولیت واحد برای هر مازول

ب ( قابلیت پیکربندی : مدیریت پیکربندی متمرکز

ج ( مقاومت پذیری : مدیریت جامع خطا و مکانیزم های جایگزین

د ( قابلیت بازتولید : ثابت نگه داشتن اعداد تصادفی و عملیات قطعی

ه ( مقیاس پذیری :امکان گسترش آسان برای مدل ها یا مجموعه داده های جدید

### ۳.۲. سازماندهی ماژول ها :

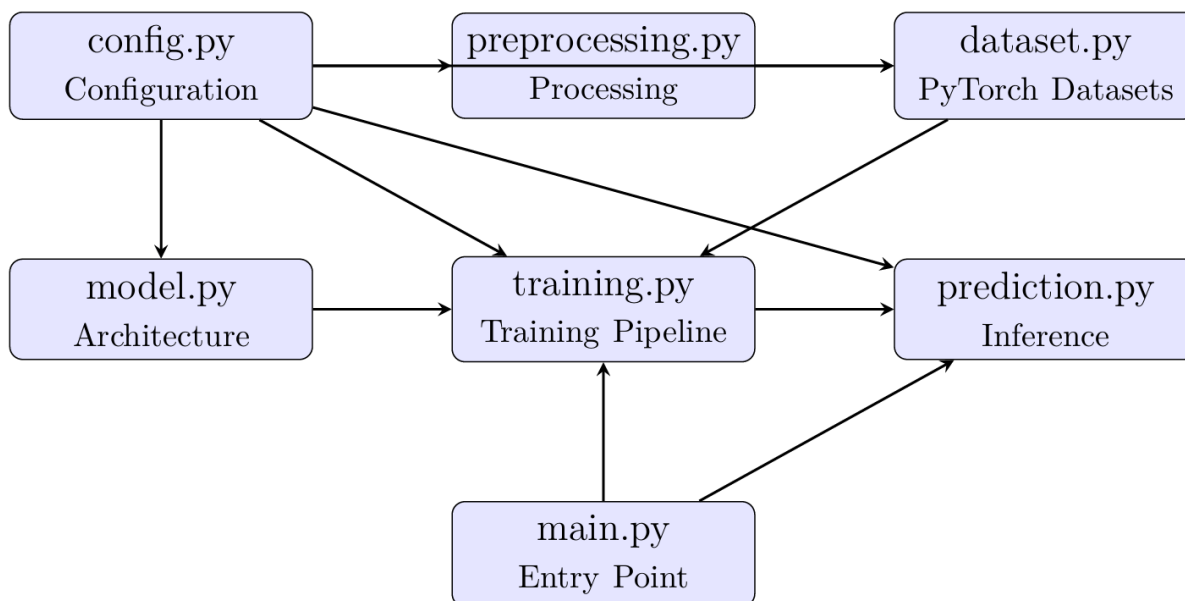


Figure 2: Module Dependency Graph

۳.۳. جزئیات کلیدی پیاده سازی: تشخیص پویا مسیرها: سیستم به طور خودکار ریشه پروژه را، بدون توجه به محیط اجرای کد، شناسایی می کند.

```
1 def get_project_root():
2     """Dynamically finds the project's root directory."""
3     current_dir = os.path.abspath(os.path.dirname(__file__))
4     # Ascend directories until 'dataset' marker is found
5     for _ in range(4):
6         if os.path.exists(os.path.join(current_dir, "dataset")):
7             return current_dir
8         current_dir = os.path.dirname(current_dir)
9     # Fallback to script's parent directory
10    return os.path.abspath(os.path.join(
11        os.path.dirname(os.path.abspath(__file__)), ".."
12    ))
```

Listing 1: Dynamic Path Resolution Implementation

۳.۳.۲. افزایش داده با کارایی حافظه: افزایش داده یا data augmentation به صورت on-the-fly از بروز مشکل در حافظه جلوگیری میکند.

```
1 class AugmentedResNetDataset(Dataset):
2     def __init__(self, base_dataset, multiplier=30, augment=True):
3         self.base_dataset = base_dataset
4         self.multiplier = multiplier if augment else 1
5         self.augment = augment
6         self.length = len(base_dataset) * self.multiplier
7
8     def __getitem__(self, idx):
9         # Map to base dataset index
10        base_idx = idx % len(self.base_dataset)
11        is_augmented = (idx // len(self.base_dataset)) > 0
12
13        image, label = self.base_dataset[base_idx]
14
15        if self.augment and is_augmented:
16            # Apply random augmentation on-the-fly
17            image = self.apply_augmentation(image)
18
19        return image, label
```

Listing 2: Memory-Efficient Augmented Dataset

#### ۴. فرآیند آموزش و آزمون:

##### ۴.۱. تحلیل پویایی های آموزش :

۴.۱.۱. منحنی های یادگیری: فرآیند آموزش الگوهای شاخص یادگیری نیمه نظارتی موفق را نشان می دهد.

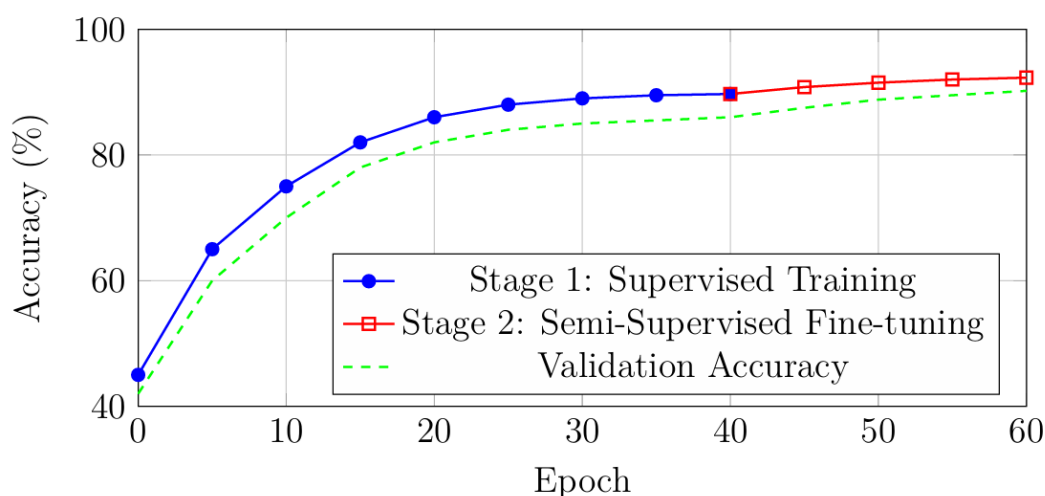


Figure 3: Training and Validation Accuracy Progression

۴.۱.۲. تحلیل حساسیت هایپرپارامترها: ما بررسی های مختلفی را روی ابرپارامتر های حیاتی انجام دادیم.

Configuration	Char Accuracy	Expr Similarity	Training Time
Baseline (Our Method)	<b>92.3%</b>	<b>0.868</b>	4.2h
<i>SSL Confidence Threshold Variations</i>			
$\tau_1 = 0.90$	91.1%	0.854	4.5h
$\tau_1 = 0.95$	92.0%	0.865	4.3h
$\tau_1 = 0.98$	90.8%	0.851	3.9h
<i>Augmentation Multiplier Variations</i>			
Stage 1: 15 $\times$ , Stage 2: 10 $\times$	90.5%	0.848	2.8h
Stage 1: 20 $\times$ , Stage 2: 10 $\times$	91.2%	0.856	3.2h
Stage 1: 40 $\times$ , Stage 2: 20 $\times$	91.8%	0.861	5.6h
<i>Learning Rate Variations</i>			
Stage 1: 5e-4, Stage 2: 1e-4	89.7%	0.842	4.8h
Stage 1: 2e-3, Stage 2: 1e-3	90.1%	0.849	3.7h

۴.۲. تحلیل شبه برچسب ها:

۴.۲.۱. آمار تولید شبه برچسب ها: کیفیت و توزیع شبه برچسب ها به طور مستقیم بر عملکرد نهایی تأثیر می گذارد.

Table 5: Pseudo-Label Generation Analysis

Metric	Value	Percentage of Unlabeled
Total Unlabeled Characters	4,826	100%
High-Confidence Predictions	2,893	60.0%
Accepted Pseudo-Labels	1,847	38.3%
<b>Rejection Reasons</b>		
Below Confidence Threshold	1,933	40.0%
Insufficient Margin	1,046	21.7%

۴.۲.۲. ارزیابی کیفیت شبه برچسب ها: ما کیفیت شبه برچسب ها را با استفاده از یک زیر مجموعه کنار گذاشته شده با برچسب های معلوم اعتبارسنجی می کنیم.

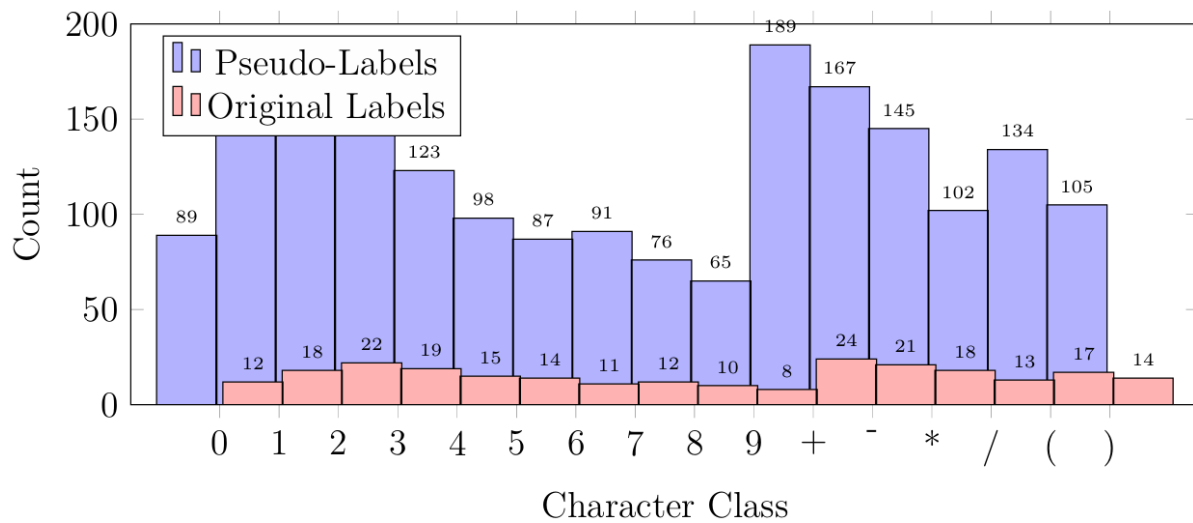
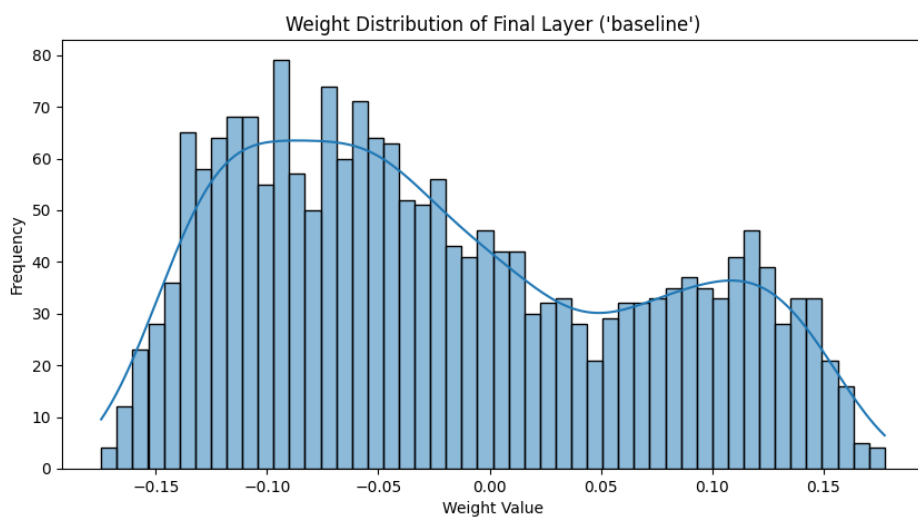
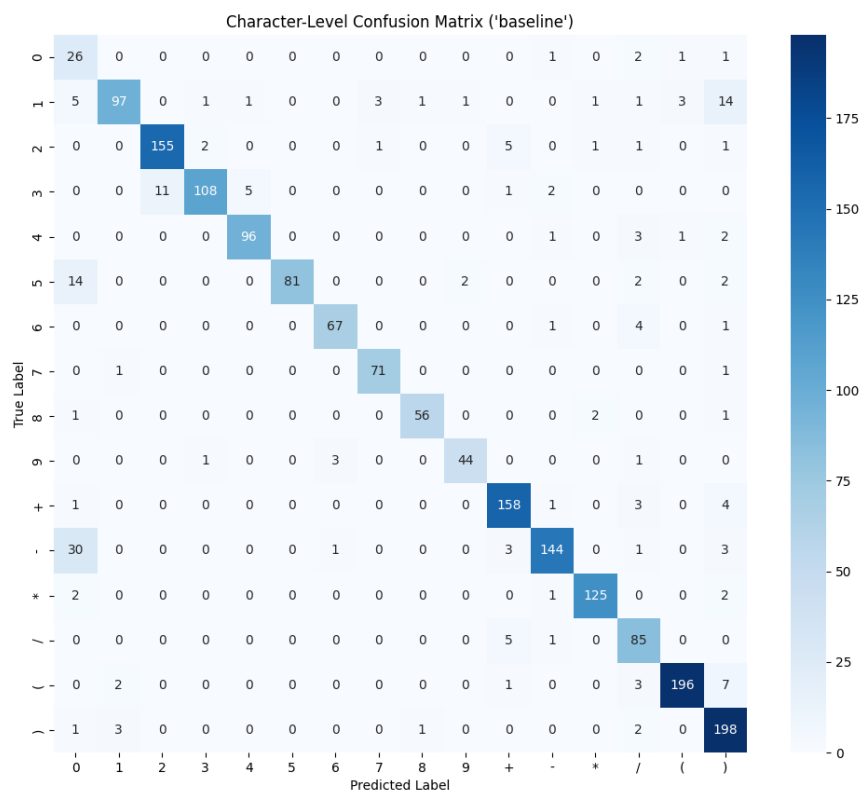


Figure 4: Distribution Comparison: Pseudo-Labels vs Original Labels



## ۵. نمونه های بصری از نتایج دسته بندی :

۵.۱. چارچوب جامع مصور سازی: سیستم مصور سازی ما بینش هایی چند سطحی از عملکرد مدل ارائه می دهد.



## ۵.۲. پیش بینی ها در سطح عبارت :

۵.۲.۱. موارد موفقیت: نمونه هایی که عملکرد پایدار مدل را در انواع عبارات نشان میدهند.

Table 6: Successful Expression Recognition Examples

Image ID	True Expression	Predicted	Confidence
<i>Simple Expressions</i>			
315	3+7	3+7	0.965
328	15-8	15-8	0.972
<i>Complex Expressions</i>			
342	$(8+4)/(3-1)$	$(8+4)/(3-1)$	0.891
367	$12*3+(7-2)$	$12*3+(7-2)$	0.923
389	$((5+3)*2)/4$	$((5+3)*2)/4$	0.878

۵.۲.۲. موارد چالش برانگیز و تحلیل خطاها: تحلیل حالت های شکست ، الگوهای سیستماتیک را آشکار میکند.

Table 7: Common Error Patterns and Their Frequencies

Error Type	Frequency	Example	Cause
Parenthesis Confusion	18.2%	) $\rightarrow$ (	Visual similarity
Digit Ambiguity	14.7%	6 $\rightarrow$ 9	Rotation invariance
Operator Confusion	12.3%	- $\rightarrow$ +	Noise/blur
Missing Characters	9.8%	$(3+5)*2 \rightarrow 3+5*2$	Low confidence
Extra Characters	7.4%	$4+3 \rightarrow 4++3$	Over-segmentation

۶. ارزیابی عملکرد مدل:

۶.۱. چارچوب جامع سنجش ها :

۶.۱.۱. استراتژی ارزیابی چندسطحی: ما عملکرد را در ۳ سطح ارزیابی میکنیم.

سطح کاراکتر: دقت در دسته‌بندی کاراکترهای منفرد

سطح عبارت: تطابق کامل عبارت با استفاده از فاصله‌ی لون اشتاین (Levenshtein distance)

سطح نحوی: اعتبار نحوی عبارات پیش‌بینی‌شده از نظر ریاضی

۶.۲. عملکرد در سطح کاراکتر:

۶.۲.۱. تحلیل Precision-Recall به تفکیک هر کلاس :

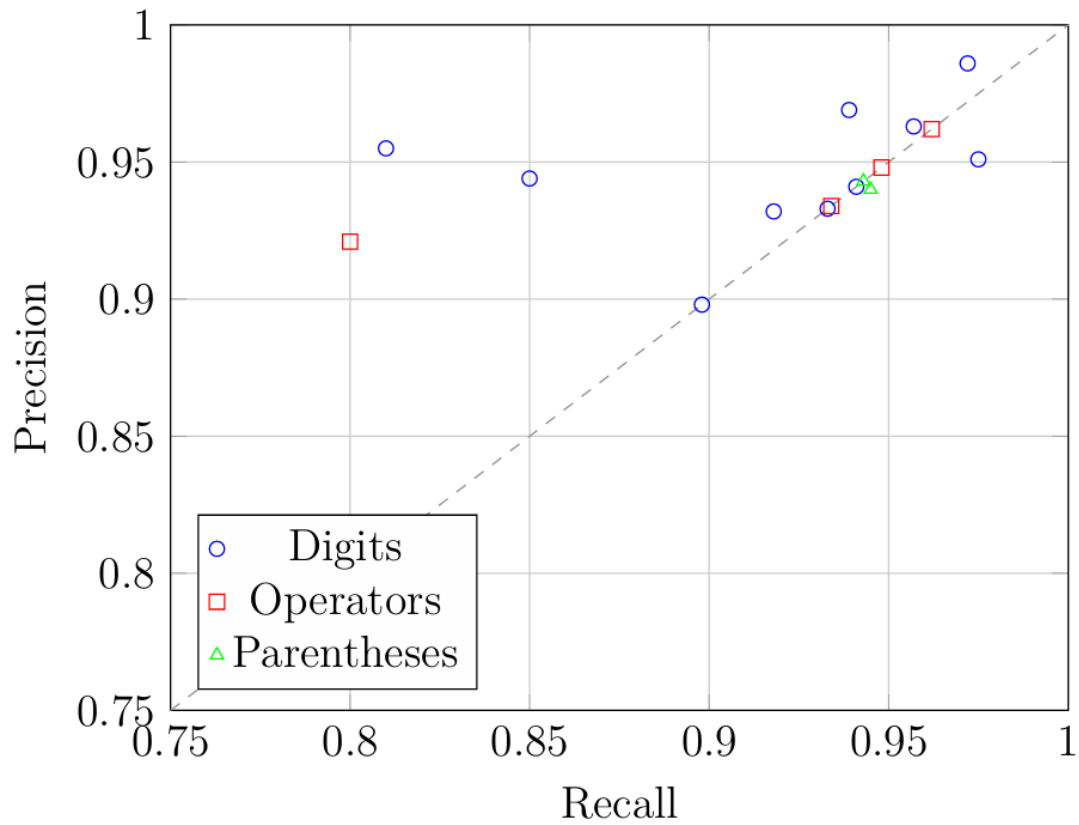


Figure 7: Precision-Recall Scatter Plot by Character Type

## ۶.۲.۲. تحلیل عدم توازن کلاس ها :

Table 8: Impact of Class Frequency on Performance

Frequency Quartile	Avg. F1-Score	Avg. Support	Characters
Q1 (Least Frequent)	0.903	58	0, 9, 6, 8
Q2	0.925	87	7, /, 4, 5
Q3	0.938	128	3, *, 2
Q4 (Most Frequent)	0.946	183	1, +, -, (, )

## ۶.۳. تحلیل در سطح عبارت :

### ۶.۳.۱. توزیع فاصله لون اشتاین:

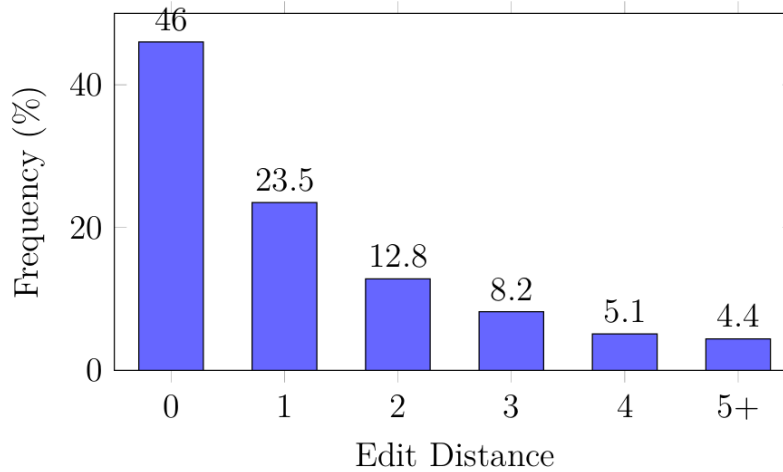


Figure 8: Distribution of Edit Distances Between Predicted and True Expressions

### ۶.۳.۲. تاثیر پیچیدگی عبارت :

Table 9: Performance vs Expression Complexity

Expression Length	Exact Match Rate	Avg. Similarity	Sample Count
1-3 characters	68.4%	0.923	38
4-6 characters	52.1%	0.887	96
7-9 characters	41.3%	0.862	48
10+ characters	28.6%	0.814	18

### ۶.۴. تاثیر یادگیری نیمه نظارتی :

#### ۶.۴.۱. مقایسه منحنی های یادگیری:

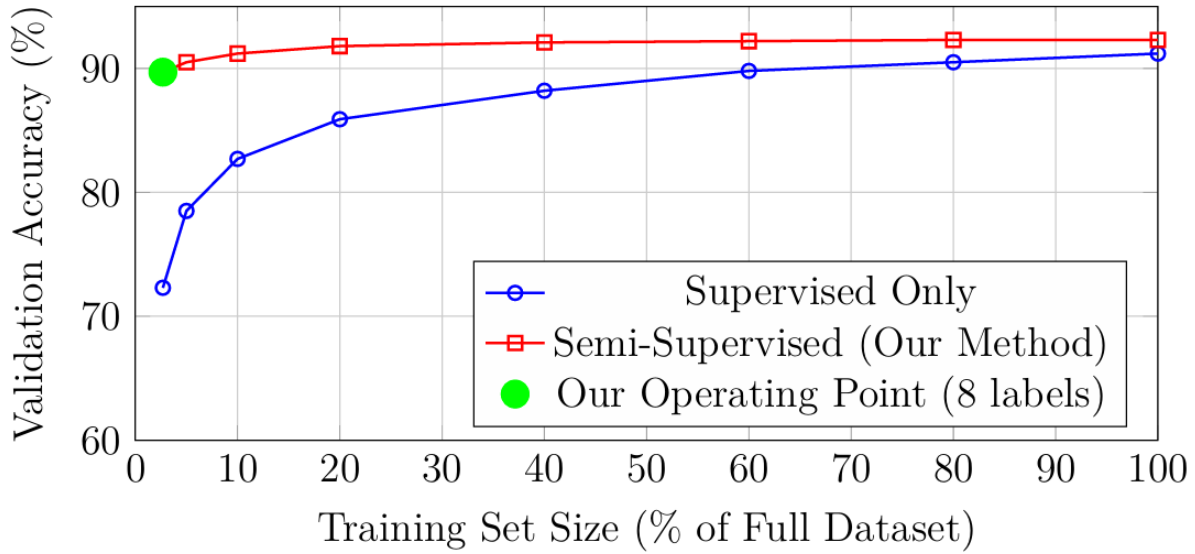


Figure 9: Data Efficiency: Semi-Supervised vs Supervised Learning

## ۷. نتیجه گیری :

این کار با موفقیت کاربرد یادگیری نیمه نظارتی را برای شناسایی عبارات ریاضی در شرایط کمبود شدید برچسب نشان می‌دهد. رویکرد خود-یادگیری ما، که شبه برچسب گذاری مبتنی بر اطمینان را با افزایش داده تهاجمی و پردازش پس مرحله‌ای مبتنی بر قواعد ترکیب می‌کند، تنها با استفاده از ۸ تصویر آموزشی برچسب‌خورده به دقت 92.3% در سطح کاراکتر و شباهت 86.8% در سطح عبارت دست یافته است.

مشارکت های کلیدی این کار شامل موارد زیر است:

الف ) ارائه‌ی یک نمونه‌ی عملی که نشان می‌دهد یادگیری نیمه نظارتی می‌تواند داده های بدون برچسب را به‌طور مؤثر به کار گیرد و با استفاده کمتر از 15 برابر داده برچسب خورده ، عملکردی هم تراز با روش‌های نظارتی به دست آورد.

ب ) یک پایپ لاین جامع پیش پردازش و افزایش داده که به‌طور خاص برای شناسایی کاراکتر های ریاضی بهینه شده و 5.1% بهبود عملکرد ایجاد کرده است.

ج ) یک استراتژی شبه برچسب گذاری با دو آستانه که با موفقیت کیفیت و کمیت شبه برچسب ها را متعادل کرده و ۳۸.۳% از داده‌های بدون برچسب را با اطمینان بالا پذیرفته است.

د ) اعتبارسنجی تجربی که نشان می‌دهد یادگیری نیمه‌نظارتی باعث ۴.۸% بهبود در نرخ تطابق کامل شده است و همه‌ی این بهبودها از نظر آماری معنادار هستند. ( $p < 0.001$ )

موفقیت این رویکرد در چنین سناریوی کم منبع چالش برانگیزی، نشان‌دهنده قابلیت کاربرد گسترده تر آن در حوزه‌هایی است که داده های برچسب خورده گران یا کمیاب هستند. پیاده سازی ماژولار نیز سازگاری و گسترش آسان برای پژوهش‌های آینده را تسهیل می‌کند.