تمرین پنجم شناسایی الگو

برای این تمرین ابتدا بخش هایی که تکمیل شده را توضیح می دهیم.

```
# ****************** START OF YOUR CODE ******************
sampled_file_paths = np.random.default_rng().choice(all_file_paths,size=self._num_support + self._num_query,replace=False)
images = [load_image(file_path) for file_path in sampled_file_paths]
images_support.extend(images[:self._num_support])
images_query.extend(images[self._num_support:])
labels_support.extend([label] * self._num_support)
labels_query.extend([label] * self._num_query)
# ****************** END OF YOUR CODE ********************
```

این بخش از کد در داخل متد __getitem__ از کلاس OmniglotDataset قرار دارد و به صورت کلی هدف آن انتخاب تصاویر مربوط به یک کلاس مشخص است.

این بخش کد درون حلقه فور است که به تعداد class_idxs اجرا میشود.

ابتدا از بین تمام تصاویر موجود در یک کلاس (در all_file_paths) تعدادی تصویر به صورت تصادفی انتخاب می شود که تعداد انتخاب ها برابر است با مجموع تعداد تصاویر support و تعداد تصاویر query است و replace برابر false است یعنی یک تصویر نمی تواند دو بار انتخاب شود.

سپس تصاویر انتخاب شده با استفاده از تابع load_image پردازش می شوند و به یک تنسور PyTorch تبدیل می شوند. در ادامه تصاویر بارگذاری شده به دو مجموعه images_support و images_query تقسیم می شوند.در نهایت label های این عکس ها نیز به دو آرایه مربوطه اضافه می شود.

```
def score(logits, labels):
  # ****************** START OF YOUR CODE ******************
    assert logits.dim() == 2
    assert labels.dim() == 1
    assert logits.shape[0] == labels.shape[0]
    y = torch.argmax(logits, dim=-1) == labels
    y = y.type(torch.float)
    return torch.mean(y).item()
  # ****************** END OF YOUR CODE ********************
```

در این تابع هدف محاسبه دقت مدل است که ابتدا از 2 بعدی بودن متغیر logits و 1 بعدی بودن labels مطمئن میشویم. همچنین تعداد نمونه ها در logits و labels باید یکسان باشد. سپس برای پیش بینی کلاس ها از هر سطر (نمونه) در logits، اندیس (کلاس) با بیشترین مقدار انتخاب می شود

و خروجی یک تنسور یک بعدی است که پیش بینی مدل برای هر نمونه را نشان می دهد. در نهایت میانگین مقادیر y محاسبه می شود که نشان دهنده درصد نمونه هایی است که پیش بینی درست داشته اند و به عنوان دقت مدل برگردانده میشود.

```
# ****************** START OF YOUR CODE ********************
in_channels = NUM_INPUT_CHANNELS
for i in range(NUM_CONV_LAYERS):
    meta_parameters[f'conv{i}'] = nn.init.xavier_uniform_(
        torch.empty(NUM_HIDDEN_CHANNELS,in_channels,KERNEL_SIZE,KERNEL_SIZE,requires_grad=True,device=DEVICE)
    )
```

به صورت کلی این بخش از کد مسئول تعریف و مقداردهی اولیه (initialization) پارامترهای مربوط به لایه های کانولوشن (Convolutional Layers) در مدل است.

متغیر in_channels تعداد کانال های ورودی به اولین لایه کانولوشن را مشخص می کند که مقدار اولیه آن برابر NUM_INPUT_CHANNELS است که بالاتر 1 در نظر گرفته شده است.

به تعداد لایه های کانولوشنی حلقه فور اجرا می شود که برای هر لایه وزن های لایه کانولوشن شماره i را در دیکشنری meta_parameters ذخیره می کند که تنسور مربوطه با ابعاد (تعداد کانال های خروجی در تعداد کانال های ورودی در ابعاد فیلتر) است که مقداردهی اولیه وزن ها با استفاده از روش Xavier Uniform انجام می شود. همچنین requires_grad=True به این معنی است که وزن ها قابل یادگیری هستند.

```
meta_parameters[f'b{i}'] = nn.init.zeros_(
    torch.empty(NUM_HIDDEN_CHANNELS,requires_grad=True,device=DEVICE)
)
in_channels = NUM_HIDDEN_CHANNELS
```

در ادامه برای تعریف بایاس های لایه کانولوشن از کد بالا استفاده می کنیم که مقدار اولیه آن ها صفر است. در اینجا بایاس های لایه کانولوشن شماره i را در دیکشنری meta_parameters ذخیره می شود که یک تنسور با اندازه تعداد کانال های خروجی است که بایاس های لایه را ذخیره می کند و همچنین قابل یادگیری هستند.

پس از تعریف لایه i تعداد کانال های خروجی لایه فعلی (NUM_HIDDEN_CHANNELS) به عنوان تعداد کانال های ورودی لایه بعدی استفاده می شود. این باعث می شود که تمامی لایه ها تعداد کانال های خروجی یکسانی داشته باشند.

```
# ****************** START OF YOUR CODE ******************
for i in range(self._num_inner_steps):
    logits = self._forward(images, parameters)
    acc = score(logits, labels)
    accuracies.append(acc)
    loss = F.cross_entropy(logits, labels)
    gradients = torch.autograd.grad(
        loss,
        inputs=list(parameters.values()),
        create_graph=train
    )
    for param_name, param_grad in zip(list(parameters.keys()), gradients):
        parameters[param_name] = parameters[param_name] - self._inner_lrs[param_name] * param_grad
```

این بخش از کد مسئول انجام حلقه داخلی (inner loop) در الگوریتم MAML است. در این حلقه پارامترهای مدل برای یک task با استفاده از داده های آموزشی آن task به روزرسانی می شوند.

حلقه فور به تعداد گام های حلقه داخلی (self._num_inner_steps) اجرا می شود و در هر مرحله با استفاده از تابع forward ورودی (تصاویر آموزشی) به مدل داده می شود سپس پیش بینی مدل (مقادیر خام خروجی از لایه نهایی) در متغیر logits قرار داده می شود. در ادامه با استفاده از تابع score پیش بینی های مدل دقت (acc) محاسبه شده و در لیست accuracies ذخیره می شود.

در ادامه خطای Cross-Entropy بین پیش بینی های مدل (logits) و برچسب های واقعی (labels) محاسبه می شود و گرادیان خطا (loss) نسبت به پارامترهای مدل (parameters.values() محاسبه می شود.

در حلقه فور بعدی برای هر پارامتر مدل مقدار فعلی پارامتر گرفته می شود و با استفاده از نرخ یادگیری داخلی گرادیان محاسبه می شود و برای به روزرسانی پارامتر ها استفاده می شود.

```
logits = self._forward(images, parameters)
acc = score(logits, labels)
accuracies.append(acc)
# ****************** END OF YOUR CODE *********************
```

پس از اتمام تمام گام های به روزرسانی داخلی، پیش بینی نهایی با استفاده از پارامترهای به روزرسانی شده انجام می شود و دقت نهایی محاسبه شده و به لیست accuracies اضافه می شود.

```
# ****************** START OF YOUR CODE ********************
parameters, accuracies_sup = self._inner_loop(images_support, labels_support, train=train)
preds_query = self._forward(images_query, parameters)
accuracy_query = score(preds_query, labels_query)
loss = F.cross_entropy(preds_query, labels_query)
accuracies_support_batch.append(accuracies_sup)
outer_loss_batch.append(loss)
accuracy_query_batch.append(accuracy_query)
# ****************** END OF YOUR CODE ********************
```

این بخش درون یک حلقه فور است که به تعداد task های درون task_batch اجرا می شود و برای هر task حلقه داخلی که در بخش قبلی توضیح دادیم را اجرا می شود و پارامتر هایی که بروزرسانی شده اند و دقت مدل که در طول فرایند حلقه داخلی محاسبه شده است را بر می گرداند.

سپس از پارامترهای به روزرسانی شده برای پیش بینی خروجی بر روی مجموعه query استفاده می کند. سپس دقت مدل با استفاده از تابع score با label های واقعی محاسبه می شود و در متغیر accuracy_query ریخته می شود. سپس خطای Cross-Entropy بین پیش بینی ها و برچسب های واقعی محاسبه می شود و به عنوان loss ذخیره می شود.

در نهایت دقت های حلقه داخلی برای داده های support به لیست مربوطه اضافه می شود.
همچنین loss و accuracy_query نیز به لیستشان اضافه می شوند.

```
# ****************** START OF YOUR CODE ********************
self._optimizer.zero_grad()
outer_loss, accuracies_support, accuracy_query = (
    self._outer_loop(task_batch, train=True)
)
outer_loss.backward()
self._optimizer.step()
# ****************** END OF YOUR CODE ********************
```

این بخش کد مسئول به روزرسانی متاپارامترها است. ابتدا قبل از شروع محاسبات جدید برای به روزرسانی پارامترها گرادیان ها صفر می شوند. در ادامه تابع self._outer_loop با task_batch اجرا می شود و خطای محاسبه شده از تمام وظایف در حلقه خارجی + دقت میانگین روی داده های support + دقت میانگین روی داده های query را بر می گرداند.

در ادامه گرادیان های مربوط به متاپارامترها محاسبه می شوند که این گرادیان ها به مدل کمک می کنند تا متاپارامترها را در جهت کاهش خطا تغییر دهد. در نهایت بهینه ساز با استفاده از گرادیان های محاسبه شده در مرحله قبل، متاپارامترهای مدل را به روزرسانی می کند.

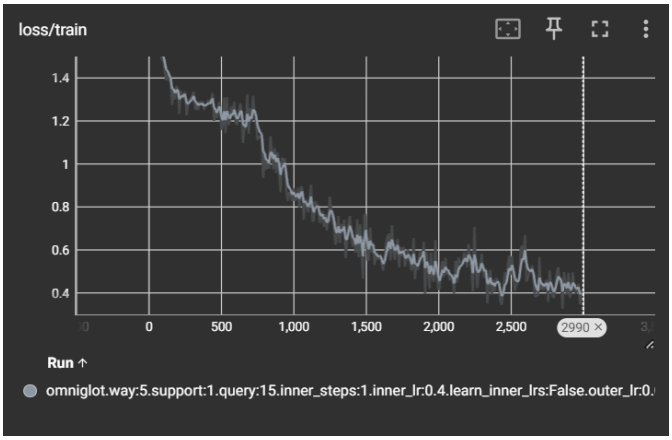
با پارامتر های پیشفرض اجرا کردیم که نتیجه به شرح زیر است:

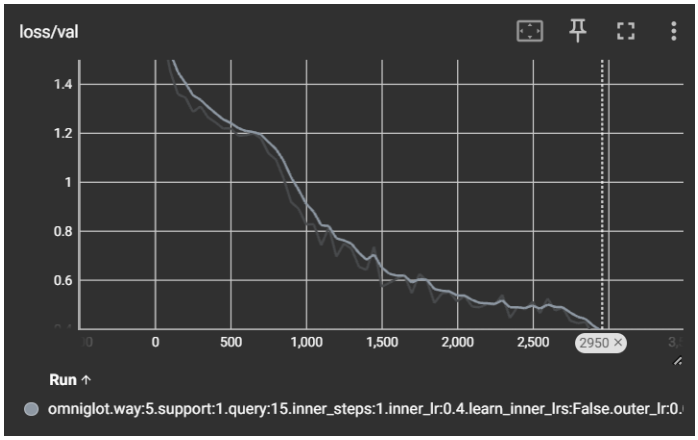
```
Validation: loss: 0.366, pre-adaptation support accuracy: 0.175, post-adaptation support accuracy: 1.000, post-adaptation query accuracy: 0.883
Iteration 2960: loss: 0.407, pre-adaptation support accuracy: 0.175, post-adaptation support accuracy: 1.000, post-adaptation query accuracy: 0.868
Iteration 2970: loss: 0.443, pre-adaptation support accuracy: 0.250, post-adaptation support accuracy: 1.000, post-adaptation query accuracy: 0.858
Iteration 2980: loss: 0.346, pre-adaptation support accuracy: 0.150, post-adaptation support accuracy: 1.000, post-adaptation query accuracy: 0.898
Iteration 2990: loss: 0.410, pre-adaptation support accuracy: 0.175, post-adaptation support accuracy: 1.000, post-adaptation query accuracy: 0.885
```

داده های تست:

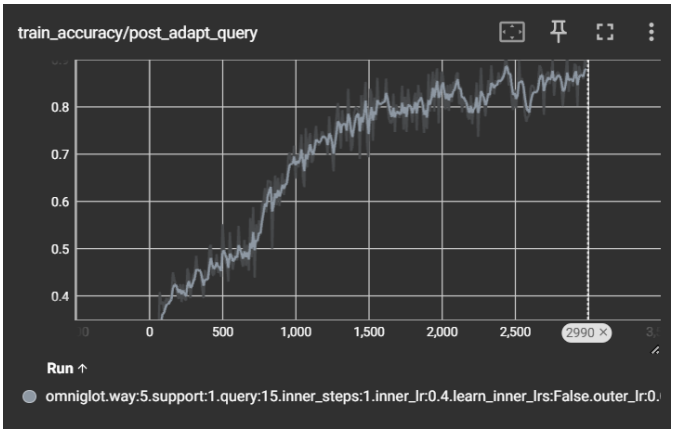
```
Testing on tasks with composition num_way=5, num_support=1, num_query=15
Accuracy over 600 test tasks: mean 0.869, 95% confidence interval 0.007
```

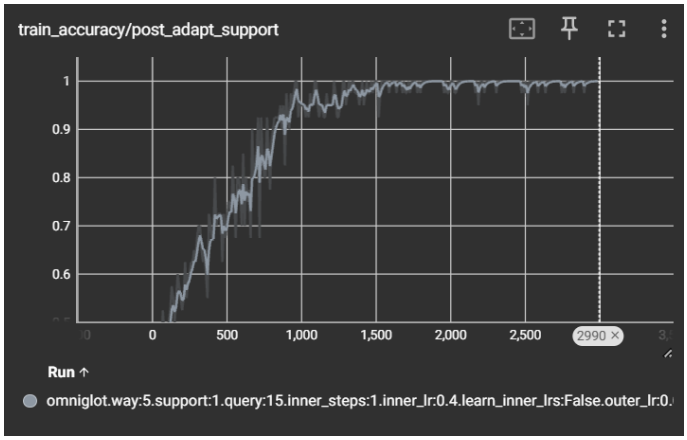
نمودار ها

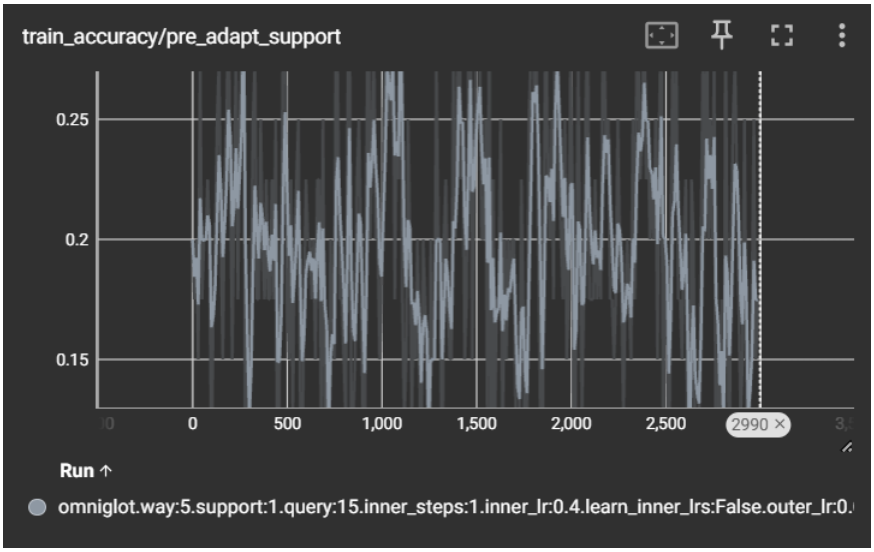




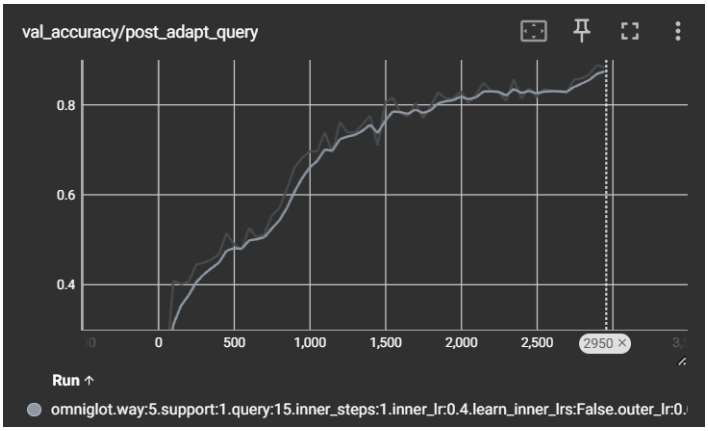
نمودار های train:

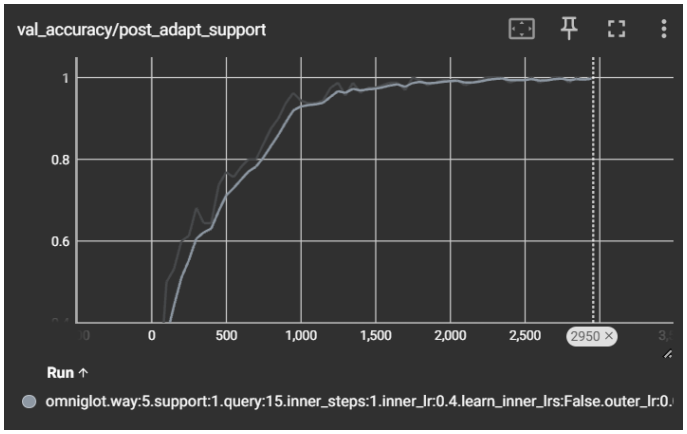


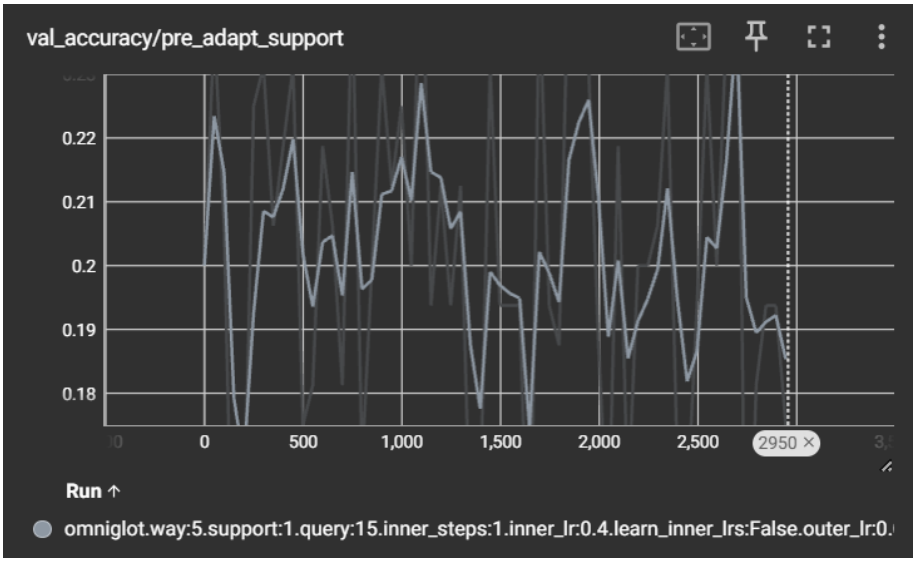




نمودار های validation







سوال 2

الف) مفاهیم ارزیابی به شرح زیر است:

Pre-adaptation support accuracy for train:

دقت بر روی مجموعه support (training subset) قبل از اینکه مدل در حین آموزش به task تطبیق پیدا کند.

Post-adaptation support accuracy for train:

دقت بر روی مجموعه support (training subset) پس از اینکه مدل در طول آموزش به task تطبیق پیدا کرد.

Post-adaptation query accuracy for train:

دقت بر روی مجموعه query (test subset) پس از اینکه مدل در حین آموزش به task تطبیق پیدا کرده است.

Pre-adaptation support accuracy for validation

دقت روی مجموعه support (validation subset) قبل از اینکه مدل در طول اعتبارسنجی به وظیفه تطبیق یابد.

Post-adaptation support accuracy for validation

دقت بر روی مجموعه support (validation subset) پس از اینکه مدل در طول اعتبارسنجی به وظیفه تطبیق پیدا کرده است.

Post-adaptation query accuracy for validation

دقت بر روی مجموعه query (test subset) پس از اینکه مدل در طول اعتبارسنجی به وظیفه تطبیق پیدا کرده است.

حال به مفهوم تطبیق (adaptation) می پردازیم که به فرآیندی اشاره دارد که در آن یک مدل یادگیری ماشین پارامترها یا نمایش های خود را برای عملکرد بهتر در یک task خاص تنظیم می کند، معمولا با استفاده از یک مجموعه کوچک از نمونه های برچسب گذاری شده (مجموعه "support") این کار صورت می گیرد.

این یک مفهوم کلیدی در meta-learning یا few-shot learning است، جایی که هدف این است که یک مدل به سرعت به وظایف جدید با حداقل داده تعمیم یابد.

ب) برای توضیح و مقایسه ی رفتار pre-adaptation support accuracy در train و validation باید ابتدا به مفهوم این دو دقت کنیم:

تعریف pre-adaptation support accuracy:

به دقت مدل روی داده های support یک task قبل از اعمال فرآیند adaptation (مانند به روزرسانی پارامترها در few-shot) اشاره دارد.

این معیار نشان می دهد که مدل، بدون اعمال تغییرات یا تنظیم مجدد برای task موردنظر، تا چه حد می تواند عملکرد خوبی داشته باشد.

رفتار در train

نحوه نمونه گیری از task ها در train

در فرآیند training ، task هایی که نمونه گیری می شوند معمولا متعلق به توزیع داده های آموزشی هستند.

مدل در طول training با این task ها آشناست و به تدریج بهینه سازی شده تا روی این توزیع عملکرد بهتری داشته باشد.

تاثیر بر pre-adaptation support accuracy

از آنجا که task های train به طور مستقیم یا غیرمستقیم در فرآیند یادگیری مدل مشارکت داشته اند، دقت pre-adaptation بالاتر است. این نشان دهنده میزان توانایی مدل برای تعمیم روی داده های مشاهده شده در training است.

رفتار در validation

نحوه نمونه گیری از task ها در validation

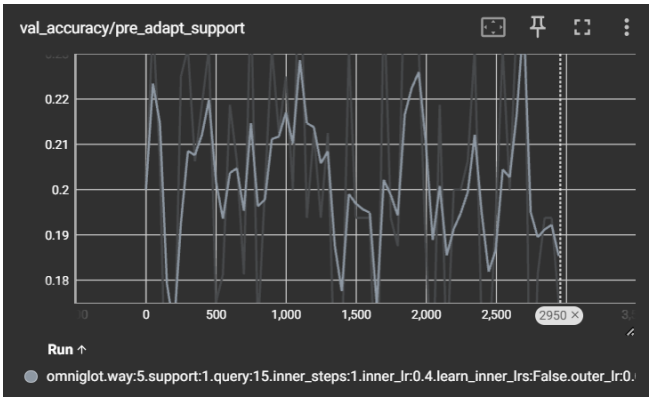
در validation ، معمولا task هایی نمونه گیری می شوند که مشابه اما جدا از توزیع داده های training هستند. این برای ارزیابی توانایی تعمیم مدل به داده های جدید است.

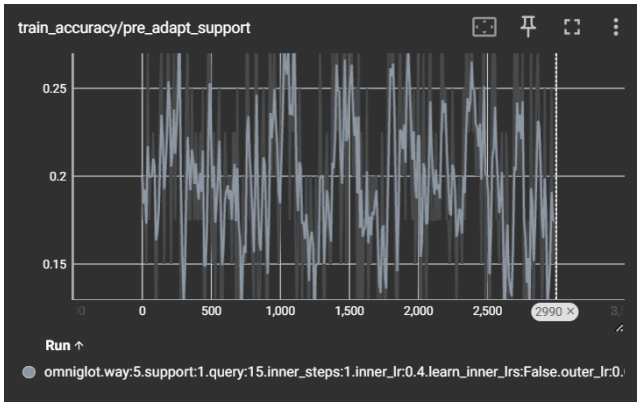
داده های validation اغلب شامل class های متفاوتی هستند که مدل در training به طور مستقیم روی آنها کار نکرده است.

تاثیر بر pre-adaptation support accuracy

از آنجا که task های validation معمولا به مدل معرفی نشده اند، pre-adaptation support accuracy در validation کمتر از train است. این اختلاف ناشی از عدم تطابق بین توزیع های training و validation است.

مقایسه کلی





دقت های train pre-adaptation support و val pre-adaption support در طول حلقه داخلی محاسبه می شوند. حلقه داخلی یک مقداردهی تصادفی از متاپارامترها را گرفته و برخی به روزرسانی های گرادیان را بر روی نمونه های برچسب گذاری شده برای هر کلاس انجام می دهد. این داده ها از هر کلاس به طور یکنواخت و تصادفی بدون جایگزینی نمونه برداری می شوند. در مسئله ما، ما یک تنظیم آموزش 5-way 1-shot داریم، که به این معنی است که ما ۱ تصویر از هر کلاس با احتمال حدود 1/5 می گیریم. این دلیل این است که دقت های train pre adapt support و val pre adapt support حدود 1/5 = 0.2 هستند.

ج) مقایسه Train Pre-Adaptation Support Accuracy و Train Post-Adaptation Support Accuracy

Train Pre-Adaptation Support Accuracy:

این دقت مربوط به عملکرد مدل روی داده های support یک task قبل از انجام فرآیند adaptation (یعنی بدون به روزرسانی پارامترها) است.

چون task های نمونه گیری شده از توزیع آموزشی هستند، مدل معمولا دقت خوبی روی آنها دارد اما نه به اندازه دقت پس از adaptation

Train Post-Adaptation Support Accuracy:

این دقت نشان دهنده عملکرد مدل روی داده های support همان task پس از اعمال adaptation است.

پس از adaptation ، مدل به صورت خاص برای همان task بهینه سازی شده است، بنابراین انتظار می رود دقت بالاتری نسبت به pre-adaptation داشته باشد.

نتیجه در Train :

دقت post-adaptation در train به طور قابل توجهی بالاتر از pre-adaptation است، زیرا فرآیند adaptation باعث می شود مدل به خوبی برای همان task تنظیم شود.

مقایسه Validation Pre-Adaptation Support Accuracy و Validation Post-Adaptation Support Accuracy

Validation Pre-Adaptation Support Accuracy:

این دقت بیانگر توانایی مدل در انجام generalization روی داده های جدیدtask ها بدون هیچ تغییری است.

معمولاً این مقدار پایین تر از pre-adaptation در train است زیرا داده های validation جدید هستند.

Validation Post-Adaptation Support Accuracy:

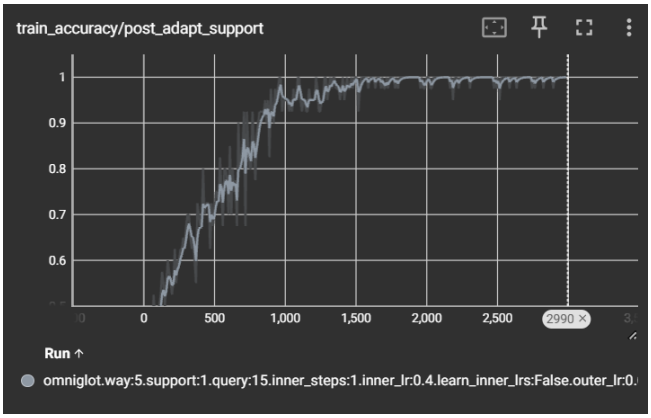
این دقت نشان دهنده عملکرد مدل پس از adaptation برای داده های validation است.

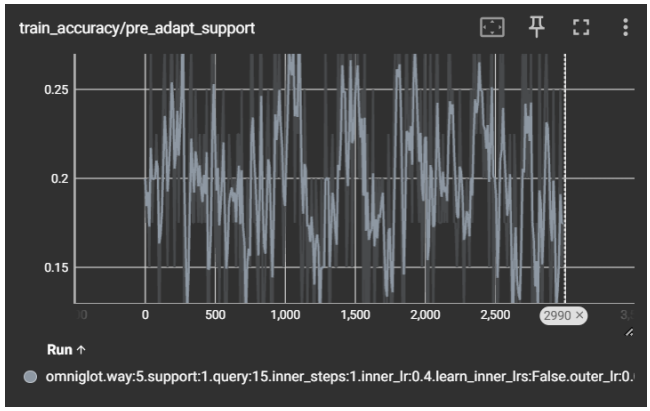
پس از فرآیند adaptation ، مدل بهینه تر عمل کرده و دقت بالاتری نسبت به pre-adaptation نشان می دهد.

نتیجه در Validation

دقت post-adaptation در validation نیز بالاتر از pre-adaptation است، اما افزایش ممکن است کمتر از train باشد زیرا task های validation می توانند تفاوت زیادی با داده های training داشته باشند.

مقایسه کلی





دقت های train pre adapt support و val pre adapt support با یک مقداردهی تصادفی از پارامترهای متا محاسبه می شوند. از سوی دیگر، دقت های train post adapt support و val post adapt support با پارامترهای متای به روز شده محاسبه می شوند که منجر به افزایش دقت در طول دوره آموزش می شود.

د) مقایسه Train Post-Adaptation Support Accuracy و Train Post-Adaptation Query Accuracy

:Train Post-Adaptation Support Accuracy

این دقت عملکرد مدل را روی داده های support همان task پس از فرآیند adaptation نشان می دهد.

این دقت معمولا بالاست، زیرا داده های support مستقیما برای به روزرسانی پارامترهای مدل در فرآیند adaptation استفاده شده اند. بنابراین، مدل به خوبی با این داده ها تطبیق پیدا کرده است.

Train Post-Adaptation Query Accuracy:

این دقت عملکرد مدل را روی داده های query همان task پس از فرآیند adaptation نشان می دهد.

داده های query در فرآیند adaptation شرکت نمی کنند و بنابراین این دقت معیار مهمی برای ارزیابی تعمیم پذیری مدل پس از adaptation است.

معمولا دقت query کمتر از دقت support است، زیرا مدل به طور خاص روی داده هایsupport تنظیم شده و تعمیم به query می تواند چالش برانگیزتر باشد.

نتیجه در Train

دقت post-adaptation support معمولا بالاتر از post-adaptation query است، زیرا داده های supportدر فرآیند بهینه سازی مشارکت داشته اند، اما داده های query به صورت غیرمستقیم از بهبودهای مدل بهره می برند.

مقایسه Validation Post-Adaptation Support Accuracy و Validation Post-Adaptation Query Accuracy

Validation Post-Adaptation Support Accuracy:

این دقت مربوط به عملکرد مدل روی داده های support برای task های validation پس از adaptation است.

دقت معمولا بالاست اما به دلیل تفاوت توزیع validation و train ، ممکن است کمتر از-train post adaptation support باشد.

Validation Post-Adaptation Query Accuracy:

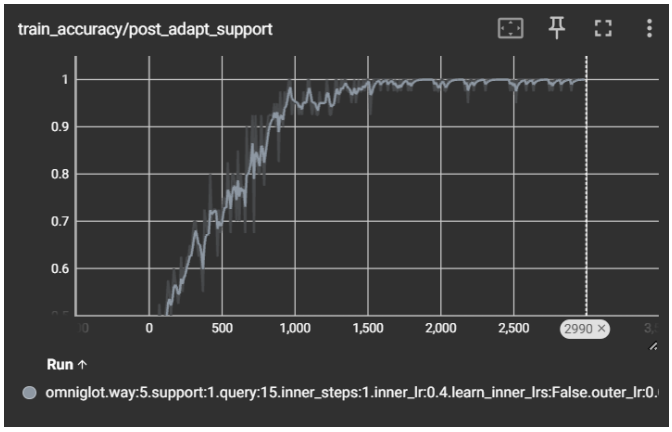
این دقت میزان تعمیم پذیری مدل روی داده های query در validation task ها پس از adaptation را نشان می دهد.

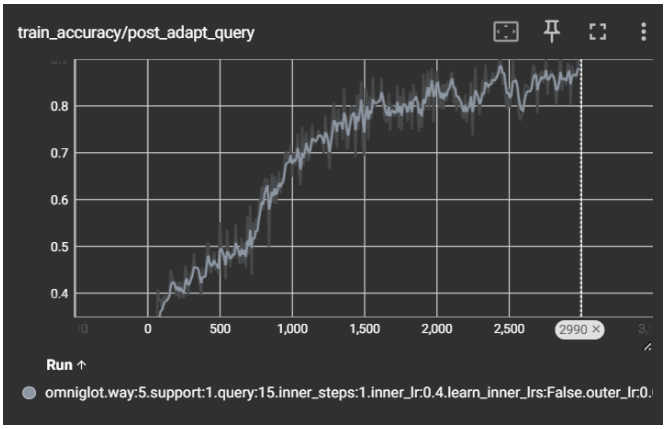
به دلیل چالش تعمیم به داده های query که در فرآیند adaptation مشارکت نداشته اند، این دقت معمولا کمتر از دقت support است.

نتیجه در Validation

- دقت post-adaptation support بیشتر از post-adaptation query است، زیرا داده های query نیازمند تعمیم هستند.
- کاهش دقت query نسبت به support در validation ممکن است بیشتر از train باشد، به دلیل تفاوت بیشتر بین داده های train و validation

مقایسه کلی



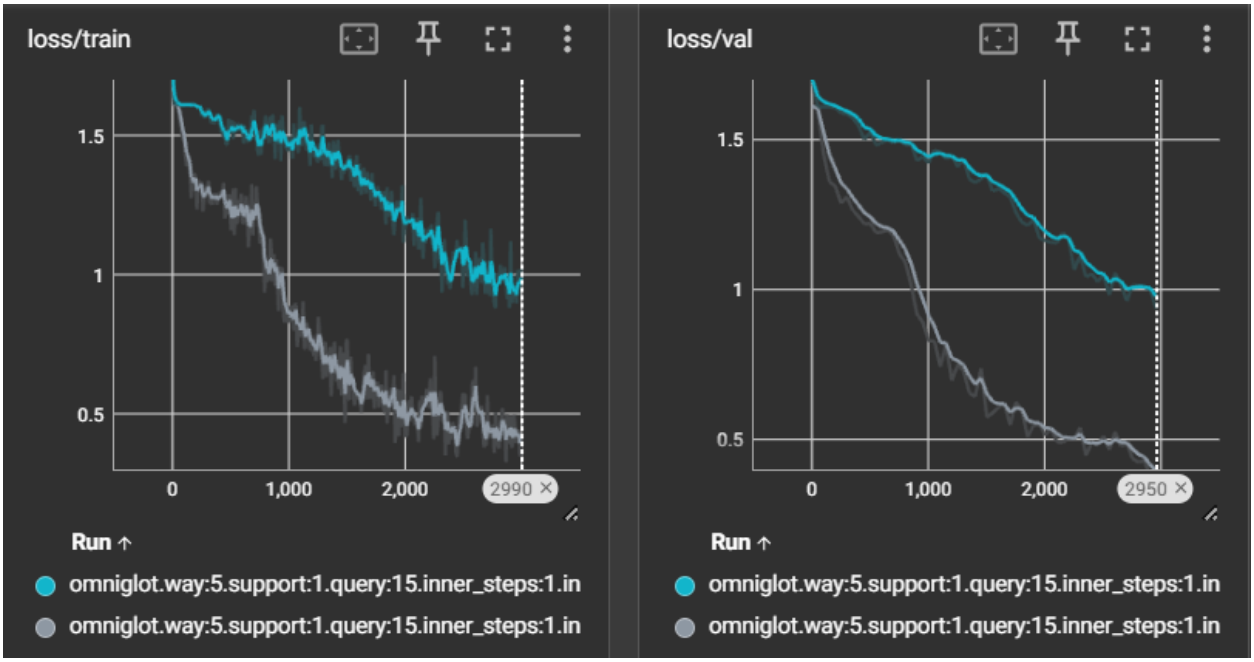


با توجه به نمودار های بالا می توانیم ببینیم که مدل ممکن است در train post adapt support و دقت های val post adapt support دچار overfitting شده باشد، اما هنوز در دقت های query ها به خوبی تعمیم می یابد.

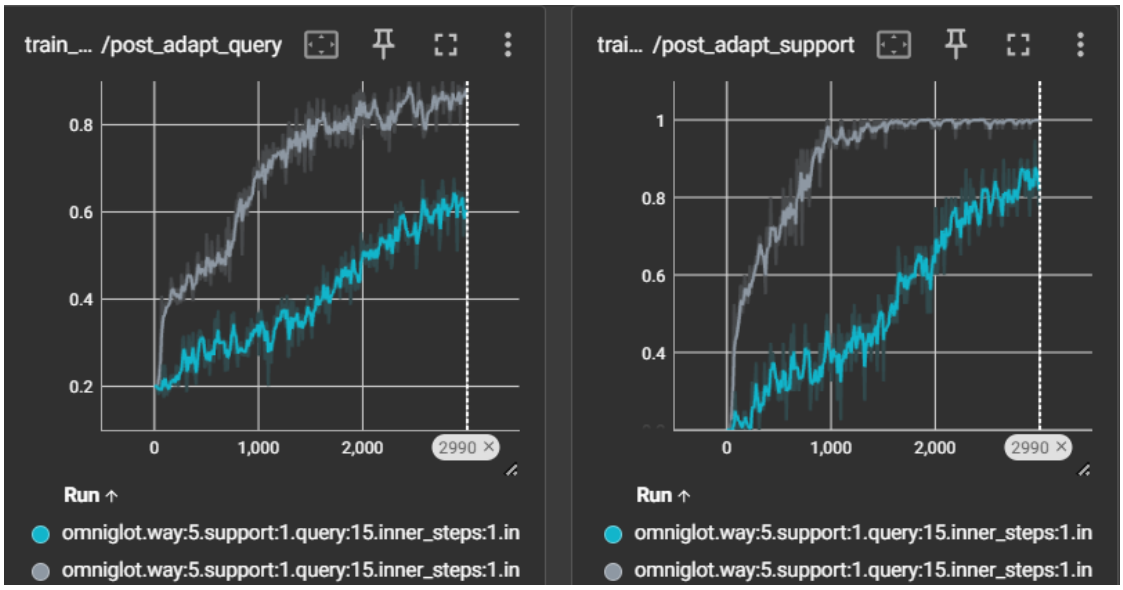
بررسی تنظیمات مختلف:

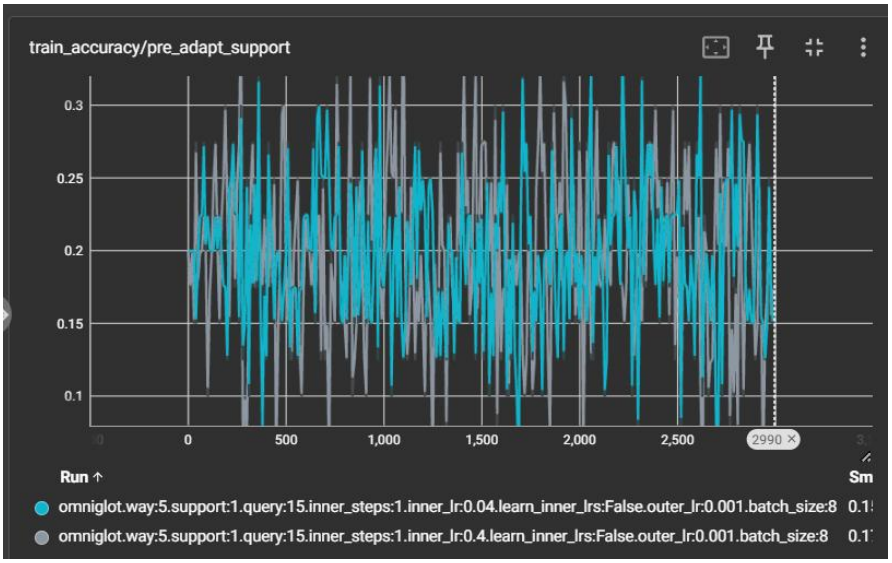
3. هایپر پارامتر inner learning rate را از 0.4 به 0.04 تغییر دادیم که نتیجه زیر حاصل شد:

```
Validation: loss: 0.936, pre-adaptation support accuracy: 0.169, post-adaptation support accuracy: 0.863, post-adaptation query accuracy: 0.617
Iteration 2960: loss: 1.018, pre-adaptation support accuracy: 0.150, post-adaptation support accuracy: 0.875, post-adaptation query accuracy: 0.543
Iteration 2970: loss: 0.990, pre-adaptation support accuracy: 0.250, post-adaptation support accuracy: 0.850, post-adaptation query accuracy: 0.563
Iteration 2980: loss: 0.951, pre-adaptation support accuracy: 0.175, post-adaptation support accuracy: 0.825, post-adaptation query accuracy: 0.633
Iteration 2990: loss: 1.020, pre-adaptation support accuracy: 0.150, post-adaptation support accuracy: 0.775, post-adaptation query accuracy: 0.592
```

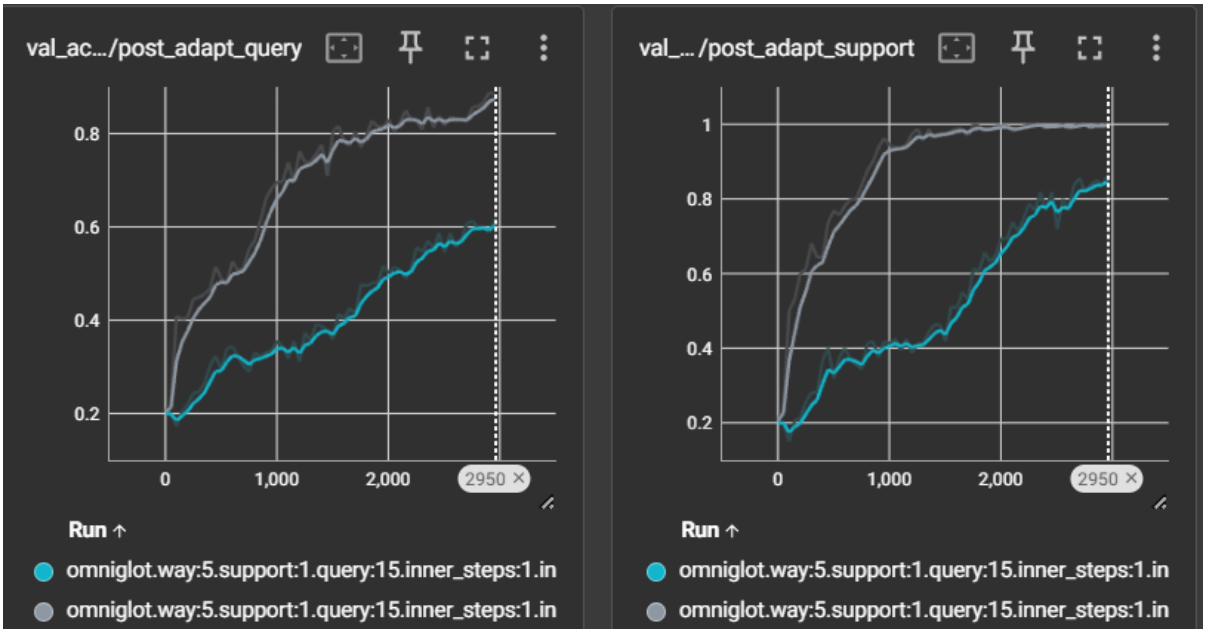


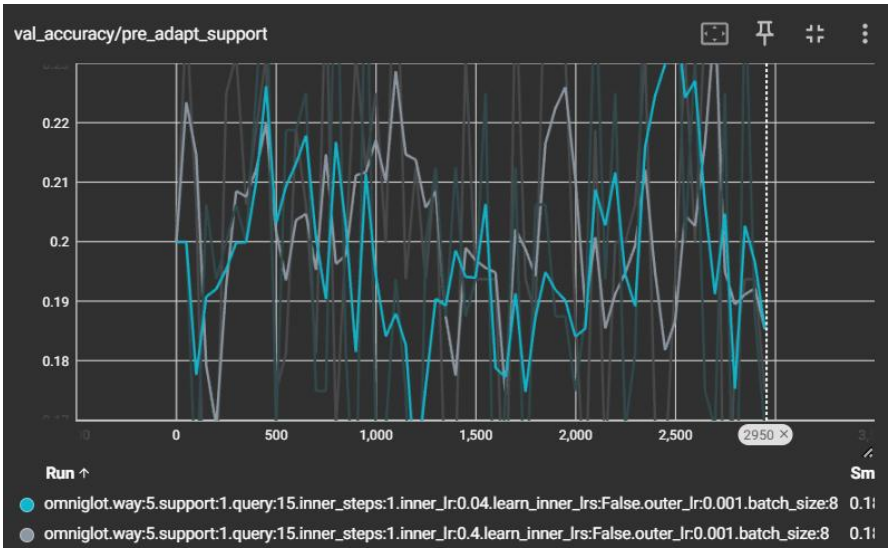
برای train:





برای validation:





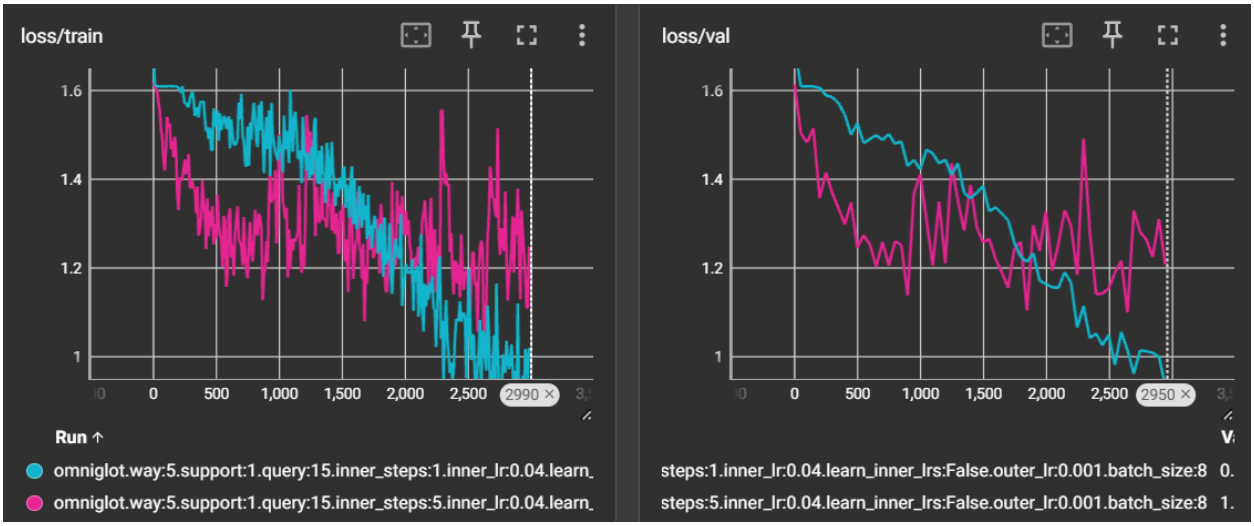
تأثیر کاهش نرخ یادگیری درونی بر بهینه سازی و تعمیم پذیری چیست؟

مقایسه نمودار ها نشان می دهد که کاهش نرخ یادگیری داخلی دقت کمتری را به همراه دارد. به نظر می رسد که کاهش نرخ یادگیری داخلی تاثیر منفی بر بهینه سازی و تعمیم پذیری (حلقه بیرونی) دارد.

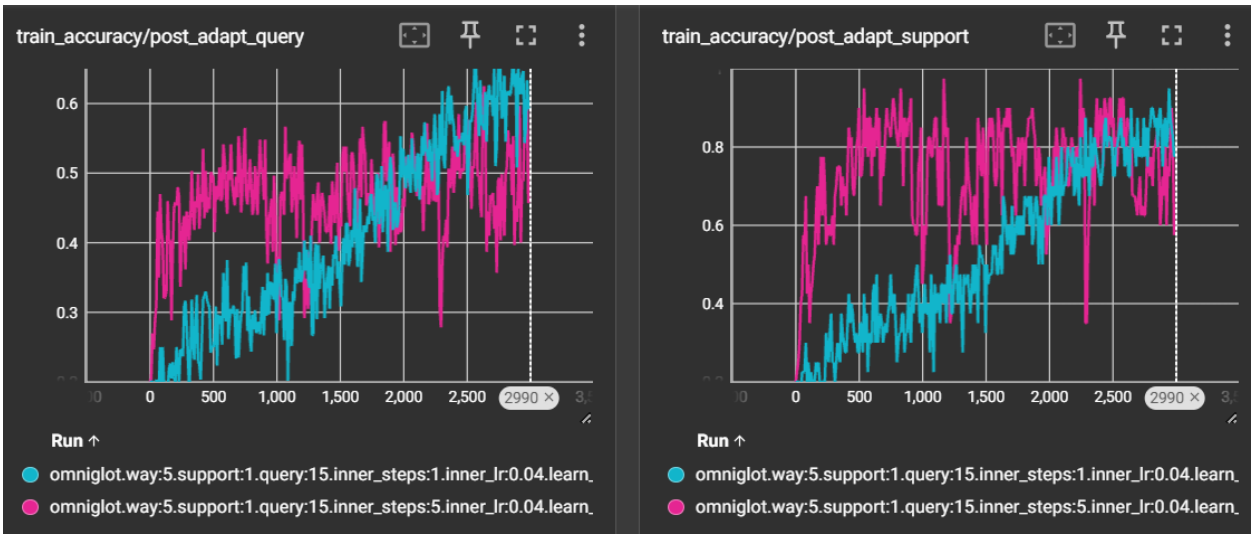
4. متغیر num_inner_steps را به 5 تغییر دادیم که به نتایج زیر رسیدیم:

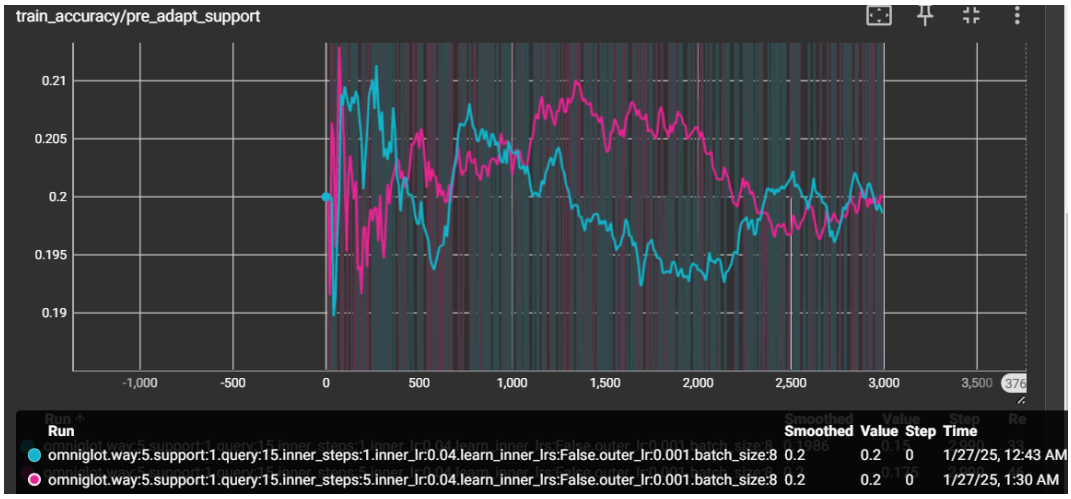
```
Validation: loss: 1.208, pre-adaptation support accuracy: 0.231, post-adaptation support accuracy: 0.825, post-adaptation query accuracy: 0.516
Iteration 2960: loss: 1.186, pre-adaptation support accuracy: 0.175, post-adaptation support accuracy: 0.800, post-adaptation query accuracy: 0.565
Iteration 2970: loss: 1.110, pre-adaptation support accuracy: 0.150, post-adaptation support accuracy: 0.900, post-adaptation query accuracy: 0.605
Iteration 2980: loss: 1.109, pre-adaptation support accuracy: 0.300, post-adaptation support accuracy: 0.825, post-adaptation query accuracy: 0.597
Iteration 2990: loss: 1.248, pre-adaptation support accuracy: 0.175, post-adaptation support accuracy: 0.575, post-adaptation query accuracy: 0.457
```

```
Testing on tasks with composition num_way=5, num_support=1, num_query=15
Accuracy over 600 test tasks: mean 0.445, 95% confidence interval 0.009
```

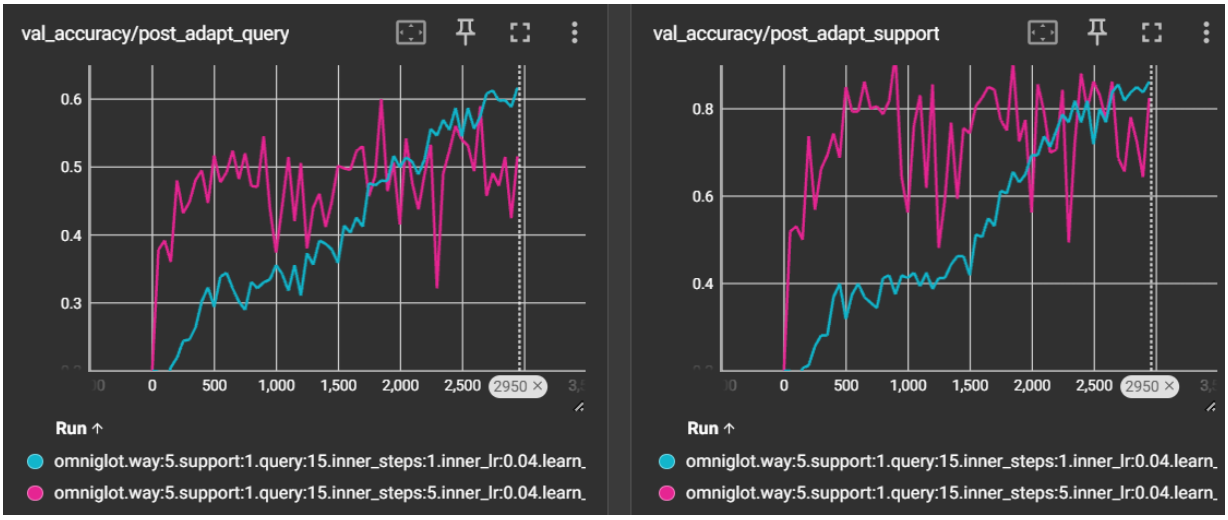


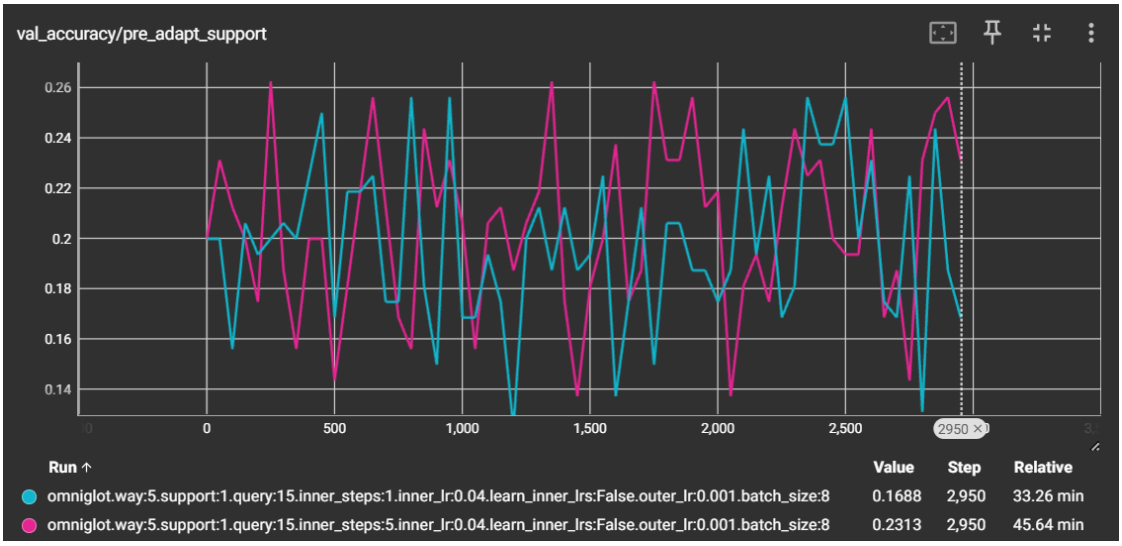
برای train:





برای validation





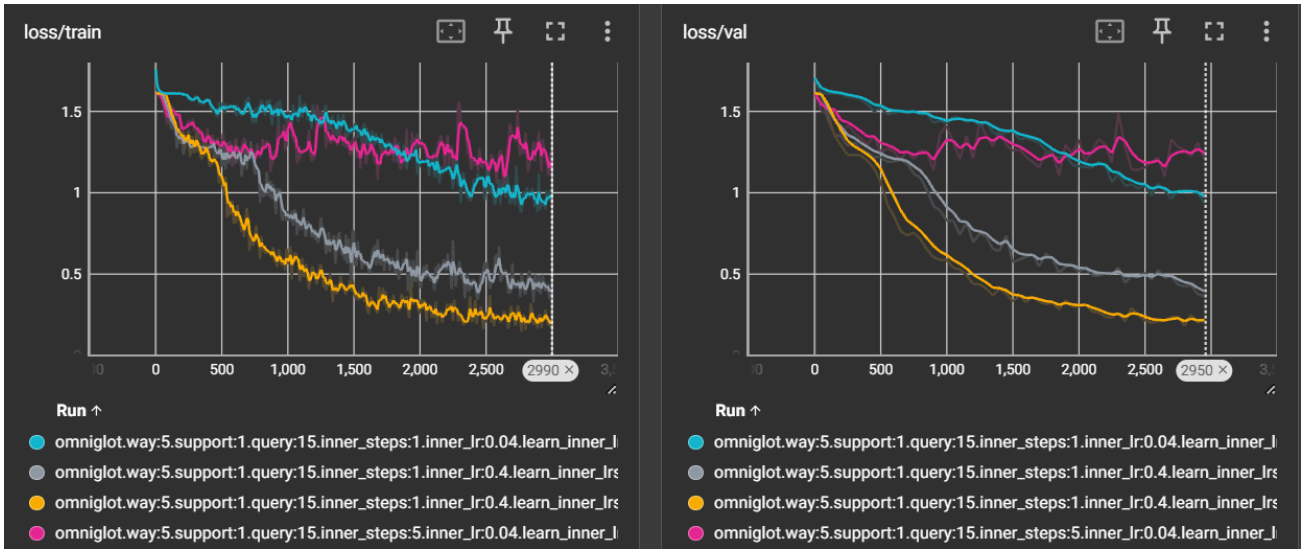
در نمودار های بالا مقایسه بین num_inner_steps 1 و 5 مقایسه شده است.

به نظر می رسد که افزایش تعداد مراحل حلقه داخلی (حلقه خارجی) تاثیر منفی بر بهینه سازی و تعمیم مدل دارد.

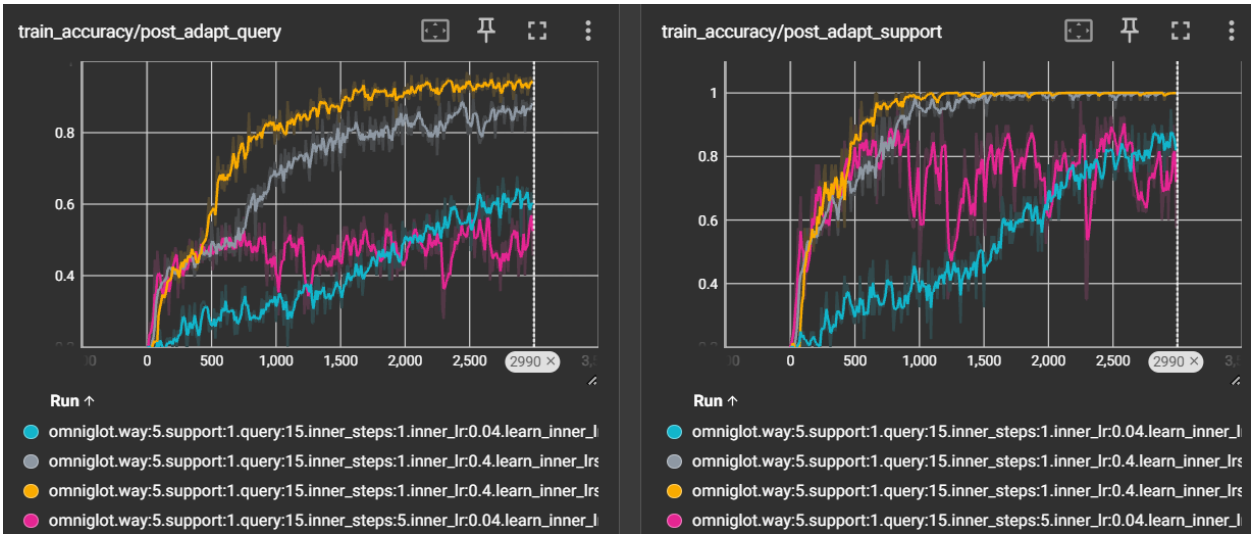
5. متغیر learn_inner_lrs را True کردیم و به نتایج زیر رسیدیم:

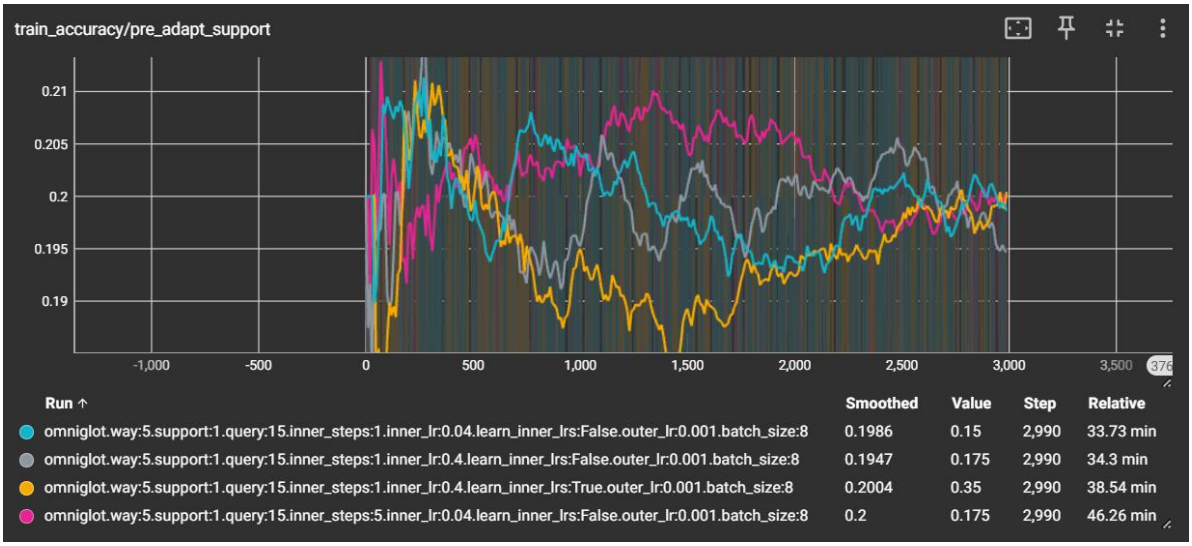
```
Iteration 2950: loss: 0.283, pre-adaptation support accuracy: 0.225, post-adaptation support accuracy: 1.000, post-adaptation query accuracy: 0.903
Validation: loss: 0.218, pre-adaptation support accuracy: 0.169, post-adaptation support accuracy: 1.000, post-adaptation query accuracy: 0.934
Iteration 2960: loss: 0.238, pre-adaptation support accuracy: 0.300, post-adaptation support accuracy: 1.000, post-adaptation query accuracy: 0.923
Iteration 2970: loss: 0.184, pre-adaptation support accuracy: 0.075, post-adaptation support accuracy: 1.000, post-adaptation query accuracy: 0.957
Iteration 2980: loss: 0.197, pre-adaptation support accuracy: 0.175, post-adaptation support accuracy: 1.000, post-adaptation query accuracy: 0.945
Iteration 2990: loss: 0.174, pre-adaptation support accuracy: 0.350, post-adaptation support accuracy: 1.000, post-adaptation query accuracy: 0.950
```

```
Testing on tasks with composition num_way=5, num_support=1, num_query=15
Accuracy over 600 test tasks: mean 0.938, 95% confidence interval 0.005
```

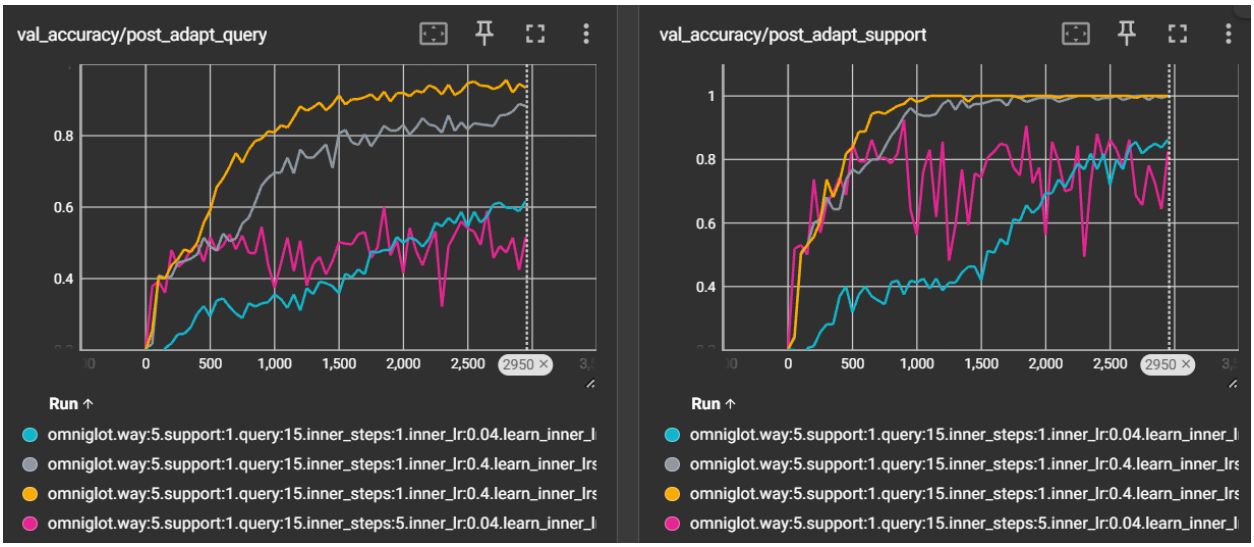


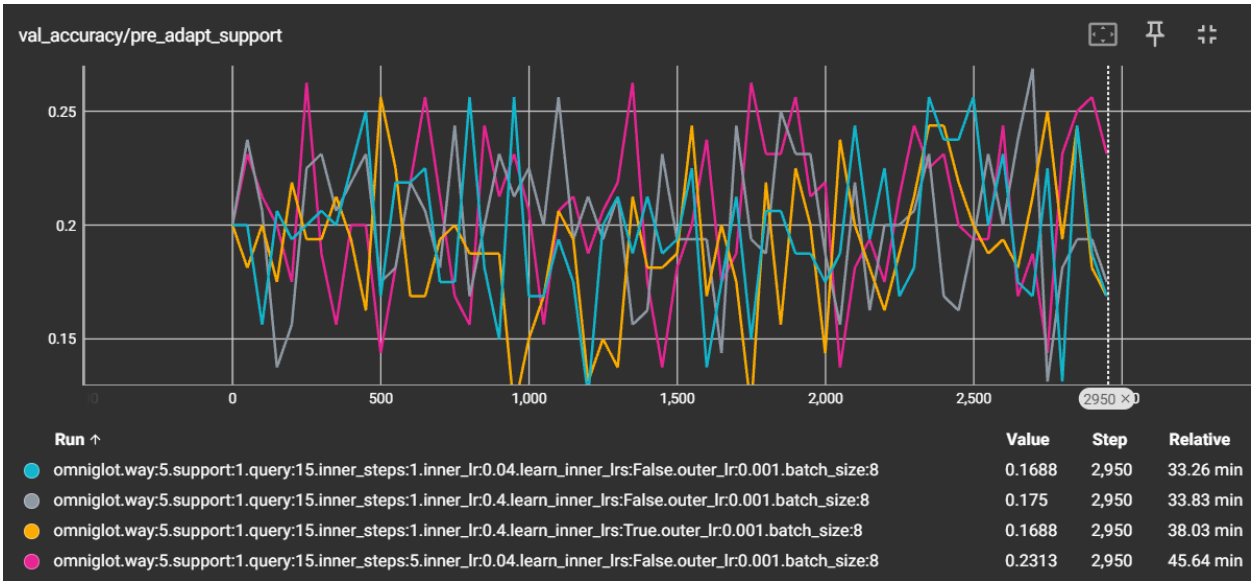
برای train:





برای validation





نمودار های بالا تغییرات دقت inner learning rates قابل یادگیری (زرد) با مدل های قبلی مقایسه می کند.

تبدیل نرخ یادگیری به یک پارامتر قابل یادگیری می تواند به بهینه سازی بهتر و تعمیم پذیری بالاتر کمک کند، زیرا مدل قادر است نرخ یادگیری را به طور داینامیک تنظیم کند. این کار می تواند سرعت همگرایی را افزایش دهد و از بهینه سازی محلی جلوگیری کند. اما ممکن است باعث نوسانات و ناپایداری در فرایند آموزش شود.

.6

```
Testing on tasks with composition num_way=5, num_support=1, num_query=10
Accuracy over 600 test tasks: mean 0.940, 95% confidence interval 0.005
```

```
Testing on tasks with composition num_way=5, num_support=2, num_query=10
Accuracy over 600 test tasks: mean 0.963, 95% confidence interval 0.004
```

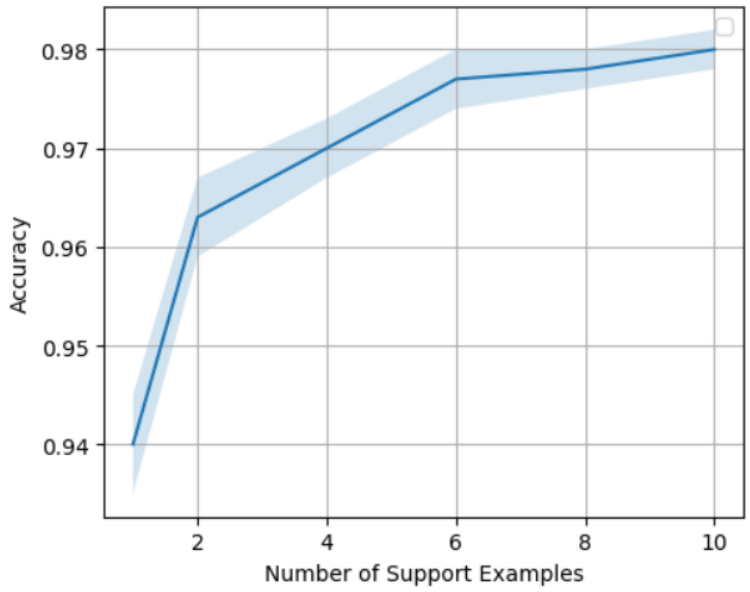
```
Testing on tasks with composition num_way=5, num_support=4, num_query=10
Accuracy over 600 test tasks: mean 0.970, 95% confidence interval 0.003
```

```
Testing on tasks with composition num_way=5, num_support=6, num_query=10
Accuracy over 600 test tasks: mean 0.977, 95% confidence interval 0.003
```

```
Testing on tasks with composition num_way=5, num_support=8, num_query=10
Accuracy over 600 test tasks: mean 0.978, 95% confidence interval 0.002
```

```
Testing on tasks with composition num_way=5, num_support=10, num_query=10
Accuracy over 600 test tasks: mean 0.980, 95% confidence interval 0.002
```

```
data = pd.DataFrame(
    [
        ["maml", 1, 0.940, 0.005],
        ["maml", 2, 0.963, 0.004],
        ["maml", 4, 0.970, 0.003],
        ["maml", 6, 0.977, 0.003],
        ["maml", 8, 0.978, 0.002],
        ["maml", 10, 0.980, 0.002],
    ],
    columns=["model", "K", "mean", "95_ci"]
)
```



تحلیل:

شروع از مقدار پایین **K=1:**

دقت مدل پایین تر است کمتر از 0.95 که نشان دهنده این است که مدل به تعداد کمی داده ساپورت حساس است و عملکرد آن محدود می شود.

افزایش اولیه **K=2** تا **K=4:**

دقت مدل به شدت افزایش یافته و به مقدار حدود 0.97 می رسد. این نشان می دهد که مدل با داده های ساپورت بیشتری می تواند ارتباطات بهتری میان کلاس ها برقرار کند.

اشباع دقت از **K=6** به بعد:
دقت مدل به حدود 0.98 می رسد و تقریبا ثابت باقی می ماند. این نشان دهنده این است که افزایش بیش ازحد داده های ساپورت (K>6) تاثیر چندانی در بهبود عملکرد ندارد.

محمد حقیقت – 403722042

برای رفع برخی ایرادات و ابهامات از Chatgpt استفاده شده است.