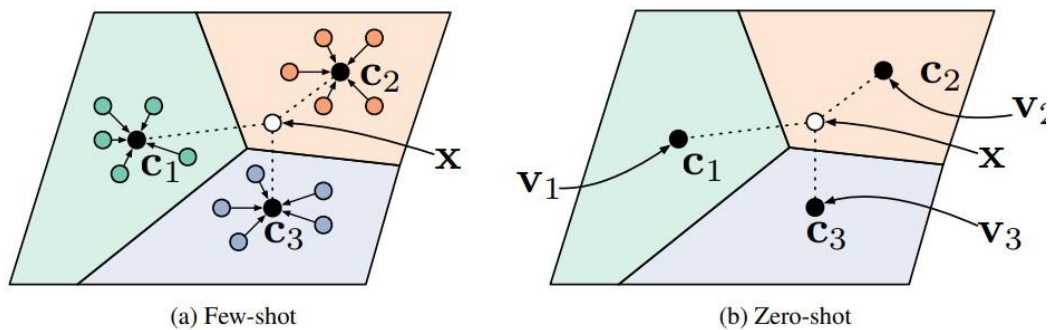


Prototypical Network

یکی از شبکه های معروفی که در مسائل few shot learning استفاده میکنند شبکه prototypical network هست به طور خلاص بخوام این شبکه رو تعریف کنم به این صورت هست که میانگین بردار فیچرهای یک کلاس یا دسته را محاسبه میکنه که برای هر کلاس با c_i نشان میدهیم منظور از i شماره کلاس است و برای image query که می خواهیم بگوییم مربوط به کدام کلاس هست فاصل اقلیدسی ان را با هر C_k از مجموعه تصویر های support محاسبه می کنیم و در یک منفی ضرب می کنیم و ماکسیمم می گیریم و image query را به آن کلاس اختصاص می دهیم. شکل زیر توصیف خوبی است از نکاتی که گفتیم:



روابط ریاضی

ما یک مجموعه support set داریم که به و به صورت $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ هست که لیبل های ما بین $\{1, 2, 3, \dots, k\}$ هست و ما مجموعه s_k را می گوییم که $y_i = k$ باشد ما c_k ها را به صورت محاسبه می کنیم:

$$c_k = \frac{1}{|S_k|} \sum_{(x_i, y_i) \in S_k} f_{\phi}(x_i)$$

و هدف ما این ست که اگر y جزو دسته k باشد احتمال زیر را بیشینه کنیم:

$$p_{\phi}(y = k | x) = \frac{\exp(-d(f_{\phi}(x), c_k))}{\sum_{k'} \exp(-d(f_{\phi}(x), c_{k'}))}$$

می تونیم تابع ضرر را به صورت زیر تعریف کنیم :

$$J(\phi) = -\log p_{\phi}(y = k | x)$$

که اگر فرمول بالا را با فرمول اول که گفتیم ترکیب کنیم تابع ضرر به صورت زیر میشود:

$$\frac{1}{N_C N_Q} \left[d(f_\phi(\mathbf{x}), \mathbf{c}_k) + \log \sum_{k'} \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_{k'})) \right]$$

که N_C برابر با تعداد کل label هایی هست که در مجموعه تصاویر query داریم و N_Q برابر با تعداد هر تصویر در آن label به خصوص.

شبه کد این الگوریتم به صورت زیر است:

Input: Training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$, where each $y_i \in \{1, \dots, K\}$. \mathcal{D}_k denotes the subset of \mathcal{D} containing all elements (\mathbf{x}_i, y_i) such that $y_i = k$.

Output: The loss J for a randomly generated training episode.

```

V ← RANDOMSAMPLE( $\{1, \dots, K\}, N_C$ )                                ▷ Select class indices for episode
for k in  $\{1, \dots, N_C\}$  do
    Sk ← RANDOMSAMPLE( $\mathcal{D}_{V_k}, N_S$ )                                ▷ Select support examples
    Qk ← RANDOMSAMPLE( $\mathcal{D}_{V_k} \setminus S_k, N_Q$ )                        ▷ Select query examples
    ck ←  $\frac{1}{N_C} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_\phi(\mathbf{x}_i)$                     ▷ Compute prototype from support examples
end for
J ← 0                                                                ▷ Initialize loss
for k in  $\{1, \dots, N_C\}$  do
    for (x, y) in Qk do
        J ← J +  $\frac{1}{N_C N_Q} \left[ d(f_\phi(\mathbf{x}), \mathbf{c}_k) + \log \sum_{k'} \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_{k'})) \right]$     ▷ Update loss
    end for
end for

```

پیاده سازی این شبکه بر روی دیتاست (5 way 5 shot) miniimagenet

در ابتدا دیتاست را از طریق colab دانلود کردم ۰.۸ را به داده های train و ۰.۲ را به داده های test اختصاص دادم و همه تصاویر را به 84 در 84 ریسایز کردم در ادامه شبکه prototype به صورت زیر پیاده سازی شده است:

```

class PrototypicalNetworks(nn.Module):
    def __init__(self, backbone:nn.Module):
        super(PrototypicalNetworks, self).__init__()
        self.backbone=backbone
    def forward(self, support_images:torch.Tensor, support_labels:torch.Tensor, query_images: torch.Tensor):

        z_support=self.backbone.forward(support_images)
        z_query=self.backbone.forward(query_images)

        n_way=len(torch.unique(support_labels))

        z_proto = torch.cat(
            [
                z_support[torch.nonzero(support_labels == label)].mean(0)
                for label in range(n_way)
            ]
        )

        dists = torch.cdist(z_query, z_proto)
        scores = -dists
        return scores

```

همانطور که از شکل بالا مشخص است دقیقا مطابق مراحل هست که ذکر کردم.

در ادامه برای ساختن یک object از شبکه به کانستراکتور backbone resnet18 را میدهم و لایه آخر این شبکه pretrained را با یک لایه flatten جایگذاری می کنیم که به صورت که بردار فیچر باشد.

ساخت مجموعه query,support

برای ساختن این دو مجموعه از کتابخانه easyfsl استفاده کردم که کد آن به صورت زیر است

```

N_WAY = 5 # Number of classes in a task
N_SHOT = 5 # Number of images per class in the support set
N_QUERY = 10 # Number of images per class in the query set
N_EVALUATION_TASKS = 100

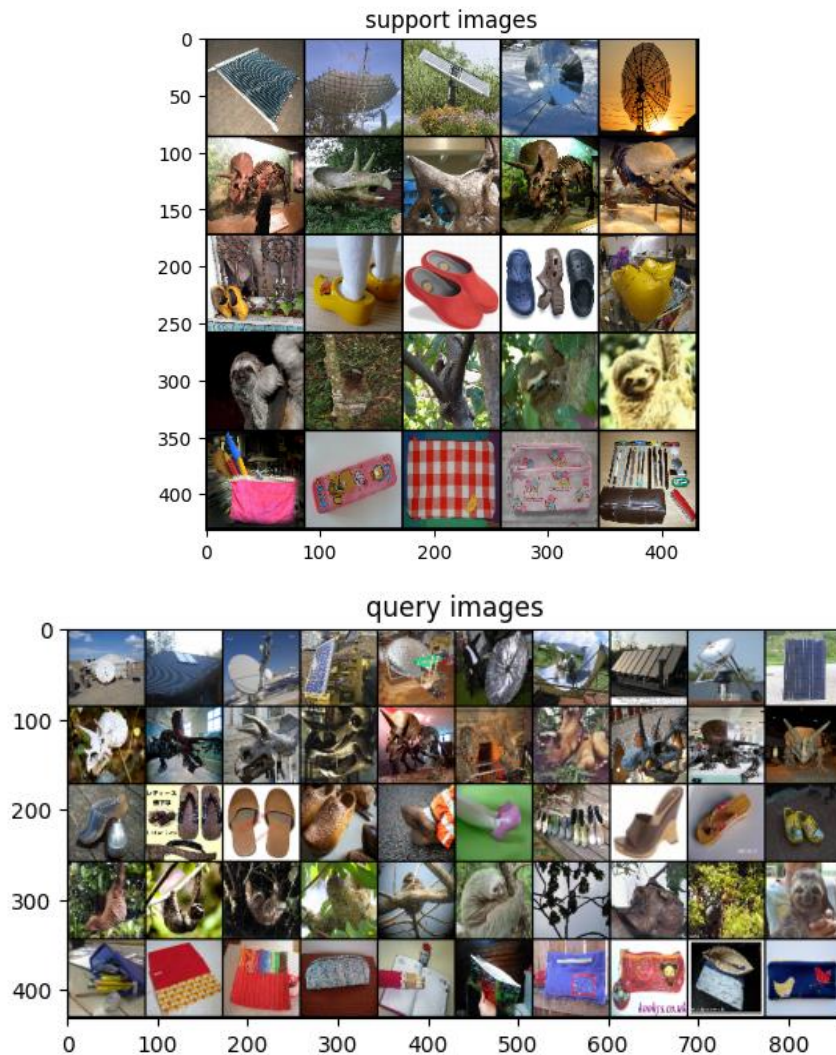
# Assign the Lambda function to get_labels
test_dataset.get_labels = lambda: [instance[1] for instance in test_dataset]

test_sampler = TaskSampler(
    test_dataset, n_way=N_WAY, n_shot=N_SHOT, n_query=N_QUERY, n_tasks=N_EVALUATION_TASKS
)

test_loader = DataLoader(
    test_dataset,
    batch_sampler=test_sampler,
    num_workers=12,
    pin_memory=True,
    collate_fn=test_sampler.episodic_collate_fn,
)

```

الان اگه ما ۱ تسک در نظر بگیریم $75 = 5 * (5+10)$ کلا ۷۵ عکس داریم یک نمونه آن به صورت زیر است:



در ابتدا می‌خواهیم بدون اینکه **train** بکنیم ببایم و بردار فیچرها را در بیاوریم و ماکس فاصله را تا **ck** برای هر تصویر کوئری تا مجموعه **support** در نظر بگیریم و بعد دقت را محاسبه کنیم و دقت برابر با 63.26 درصد شد

Train کردن مدل

در ابتدا برای داده‌های آموزشی هم باید مجموعه **support** و **query** رو هم درست کرد. در پیاده‌سازی از **loss** همان **cross entropy** استفاده کرده. و سپس بعد از **train** کردن مدل به دقت 75.10 درصد رسیدیم یعنی حدود ۱۲ درصد نسبت به قبل از **train** پیشرفت داشتیم. 75.1%