Few-Shot Model Performance Evaluation

Evaluating the performance of a few-shot model in image classification involves assessing how well the model can learn from a limited number of examples (few shots) and make accurate predictions. Here are some common evaluation techniques for few-shot image classification models:

- **Accuracy**
- **Top-k Accuracy**
- **Confusion Matrix**
- **Precision, Recall, and F1 Score**

Accuracy:

The most straightforward evaluation metric is accuracy, which measures the percentage of correctly classified images in the test dataset. However, accuracy may not be the best metric for few-shot learning as models might perform well on classes with more examples or instances in the dataset but poorly on classes with a relatively small number of examples.

In our 5-way 5-shot plant leaf disease classification using prototypical network, we use the 'evaluate()' method from "easyfsl" repo to calculate the model's accuracy.

```
[15]
evaluate(test_loader)
                                                                    Python

... 100%|██████████| 100/100 [00:02<00:00, 36.57it/s]
    Model tested on 100 tasks. Accuracy: 80.06%
```

As we can see from the image in our test case the accuracy of our 5-way 5-shot model was 80.06%

Top-k Accuracy:

In addition to standard accuracy, we also use top-k accuracy, which measures the percentage of test samples for which the correct label is within the top-k predicted labels. This can be more forgiving than standard accuracy, especially in cases where the model can identify the correct class but is uncertain about the exact label.

```python
def top_k_accuracy(query_labels, predicted_scores, k=1):
    # Sort predicted scores in descending order and get the indices of the top-k predictions
    _, topk_indices = torch.topk(predicted_scores, k, dim=1, largest=True, sorted=True)

    correct_predictions = 0
    total_predictions = query_labels.size(0)

    for i in range(total_predictions):
        if query_labels[i] in topk_indices[i]:
            correct_predictions += 1

    accuracy = correct_predictions / total_predictions
    return accuracy
[16]                                                                Python
```

```python
def evaluate_with_topk(data_loader, topk=1):
    total_predictions = 0
    correct_predictions = 0

    model.eval()
    with torch.no_grad():
        for episode_index, (
            support_images,
            support_labels,
            query_images,
            query_labels,
            class_ids,
        ) in tqdm(enumerate(data_loader), total=len(data_loader)):

            classification_scores = model(
                support_images.cuda(), support_labels.cuda(), query_images.cuda()
            )

            _, predicted_labels = torch.max(classification_scores.data, 1)

            correct, total = evaluate_on_one_task(
                support_images, support_labels, query_images, query_labels
            )

            total_predictions += total
            correct_predictions += correct

    accuracy = correct_predictions / total_predictions
    topk_accuracy = top_k_accuracy(query_labels, classification_scores, topk)

    # Calculate top-3 accuracy
    top3_accuracy = top_k_accuracy(query_labels, classification_scores, 3)

    return accuracy, topk_accuracy, top3_accuracy

# Evaluate the model with top-1, top-3, and top-5 accuracy
accuracy, top1_accuracy, top3_accuracy = evaluate_with_topk(test_loader, topk=5)
print(f"Top-1 Accuracy: {accuracy * 100:.2f}%")
print(f"Top-3 Accuracy: {top3_accuracy * 100:.2f}%")
print(f"Top-5 Accuracy: {top1_accuracy * 100:.2f}%")
```

```
100%|██████████| 100/100 [00:03<00:00, 27.52it/s]
Top-1 Accuracy: 79.58%
Top-3 Accuracy: 98.00%
Top-5 Accuracy: 100.00%
```

In this instance, top-1, 3, 5 accuracies were 79.58%, 98%, 100% which indicates for each example, the model's correct prediction is within the top 1, 3, 5 predicted classes approximately 79.58%, 98%, 100% of the time.

Confusion Matrix:

A confusion matrix provides a more detailed view of the model's performance. It shows how many samples were classified correctly and how many were misclassified for each class. This is particularly useful for understanding which classes are challenging for the few-shot model.

```python
from sklearn.metrics import confusion_matrix

def create_confusion_matrix(data_loader):
    true_labels = []
    predicted_labels = []

    model.eval()
    with torch.no_grad():
        for episode_index, (
            support_images,
            support_labels,
            query_images,
            query_labels,
            _,
        ) in tqdm(enumerate(data_loader), total=len(data_loader)):

            classification_scores = model(
                support_images.cuda(), support_labels.cuda(), query_images.cuda()
            )

            _, predicted = torch.max(classification_scores.data, 1)

            true_labels.extend(query_labels.cpu().numpy())
            predicted_labels.extend(predicted.cpu().numpy())

    cm = confusion_matrix(true_labels, predicted_labels)
    return cm

confusion_matrix_result = create_confusion_matrix(test_loader)
print("Confusion Matrix:")
print(confusion_matrix_result)
```

```
100%|██████████| 100/100 [00:02<00:00, 36.43it/s]
Confusion Matrix:
[[845  80  26  28  21]
 [ 68 823  36  28  45]
 [ 29  69 795  51  56]
 [ 62  57  33 780  68]
 [ 38  45  47  82 788]]
```

In our model, the confusion matrix is a 5x5 matrix representing the results of a classification task with five classes. Each row in the matrix corresponds to the actual or true class, while each column corresponds to the predicted class. So, the value 845 of row 0 and column 0 represents 812 instances of class 0 that were correctly classified as class 0. The value 54 of row 0 and column 1 represents that 54 instances of class 0 were incorrectly classified as class 1. And this pattern continues for the other rows and columns.

Precision, Recall, and F1 Score:

These three metrics are helpful when you want to evaluate the model's performance on individual classes within the few-shot setting.

Precision measures the percentage of true positives among all predicted positives. It answers the question: "Of all the instances that the model predicted as positive, how many were actually positive?"

Recall measures the percentage of true positives among all actual positives. It answers the question: "Of all the actual positive instances, how many did the model correctly identify as positive?"

The **F1 score** is the harmonic mean of precision and recall. These metrics are especially relevant when classes are imbalanced.

$$Precision = True\ Positives\ /\ (True\ Positives + False\ Positives)$$

$$Recall = True\ Positives\ /\ (True\ Positives + False\ Negatives)$$

$$F1\ Score = 2 * (Precision * Recall)\ /\ (Precision + Recall)$$

```python
from sklearn.metrics import precision_score, recall_score, f1_score

true_labels = []
predicted_labels = []

model.eval()  # Set the model in evaluation mode
with torch.no_grad():
    for episode_index, (support_images, support_labels, query_images, query_labels, _) in tqdm(enumerate(test_loader), total=len(test_loader)):
        predicted = model(support_images.cuda(), support_labels.cuda(), query_images.cuda()).detach()
        _, predicted_labels_batch = torch.max(predicted.data, 1)
        true_labels.extend(query_labels.cpu().numpy())
        predicted_labels.extend(predicted_labels_batch.cpu().numpy())

true_labels = np.array(true_labels)
predicted_labels = np.array(predicted_labels)

precision = precision_score(true_labels, predicted_labels, average='weighted')
recall = recall_score(true_labels, predicted_labels, average='weighted')
f1 = f1_score(true_labels, predicted_labels, average='weighted')

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
[19]                                                                  Python
100%|██████████| 100/100 [00:02<00:00, 36.23it/s]
Precision: 0.80
Recall: 0.80
F1-Score: 0.80
```

Our model's Precision, Recall, and F1 score were all 0.80.

A **precision** of 0.80 means that out of all the positive predictions made by the model, 80% were correct.

Recall of 0.80 means that the model correctly identified 80% of all actual positive instances in the dataset.

An **F1-score** of 0.80 indicates that the model achieves a good balance between precision and recall. This suggests that the model is effective at making accurate positive predictions while also capturing a substantial portion of the actual positive instances in the dataset.

Dataset Structure:

```
Dataset
├── images_background
|   ├── Apple
|   |   ├── Apple__Apple_scab
|   |   ├── Apple__Black_rot
|   |   ├── Apple__Cedar_apple_rust
|   |   └── Apple__healthy
|   ├── Corn
|   |   ├── Corn_(maize)__Cercospora_leaf_spot Gray_leaf_spot
|   |   ├── Corn_(maize)__Common_rust_
|   |   ├── Corn_(maize)__Northern_Leaf_Blight
|   |   └── Corn_(maize)__healthy
|   ├── Grape
|   |   ├── Grape__Black_rot
|   |   ├── Grape__Esca_(Black_Measles)
|   |   ├── Grape__Leaf_blight_(Isariopsis_Leaf_Spot)
|   |   └── Grape__healthy
|   ├── Potato
|   |   ├── Potato__Early_blight
|   |   ├── Potato__Late_blight
|   |   └── Potato__healthy
|   ├── Strawberry
|   |   ├── Strawberry__Leaf_scorch
|   |   ├── Strawberry___Leaf_scorch
|   |   ├── Strawberry___healthy
|   |   └── Strawberry__healthy
|   └── Tomato
|       ├── Tomato_Bacterial_spot
|       ├── Tomato_Early_blight
|       ├── Tomato_Late_blight
|       ├── Tomato_Leaf_Mold
|       ├── Tomato_Septoria_leaf_spot
|       ├── Tomato_Spider_mites_Two_spotted_spider_mite
|       ├── Tomato__Bacterial_spot
|       ├── Tomato__Early_blight
|       ├── Tomato__Late_blight
|       ├── Tomato__Leaf_Mold
|       ├── Tomato__Septoria_leaf_spot
|       ├── Tomato__Spider_mites Two-spotted_spider_mite
|       ├── Tomato__Target_Spot
|       ├── Tomato__Tomato_Yellow_Leaf_Curl_Virus
|       ├── Tomato__Tomato_mosaic_virus
|       ├── Tomato__healthy
|       └── Tomato_healthy
└── images_evaluation
    ├── Blueberry
    |   └── Blueberry__healthy
    ├── Cherry
    |   ├── Cherry_(including_sour)__Powdery_mildew
    |   └── Cherry_(including_sour)__healthy
    ├── Orange
    |   └── Orange__Haunglongbing_(Citrus_greening)
    ├── Papper
    |   ├── Pepper__Bacterial_spot
    |   └── Pepper__healthy
    ├── Peach
    |   ├── Peach__Bacterial_spot
    |   └── Peach__healthy
    ├── Raspberry
    |   └── Raspberry__healthy
    ├── Soybean
    |   └── Soybean__healthy
    └── Squash
        └── Squash__Powdery_mildew
```