Towards Automated Petrography

Isai Daniel Chacón*
Universidad de los Andes, Colombia

Paola Ruiz Puentes*
Universidad de los Andes, Colombia

Jillian Pearse
California State University, Long Beach

Pablo Arbeláez
Universidad de los Andes, Colombia

1 Database organization

The dataset introduced in the main paper is publicly available on Kaggle. The root directory contains two folders, $GMAS\_LOTE$ and $GMAS\_LOTE2$, which together include 580 thin-section samples (383 in the former, 197 in the latter). Each thin section is stored as a subfolder that contains multiple grain-level annotations. Each folder comprises three files at the grain level: PPL-0.png and XPL-0.png images captured under plane- and cross-polarized light, respectively, and an annotation file named $center\_component.csv$. This CSV file includes detailed attributes for each grain: $Multiclass\_label\_name$, $Multiclass\_label$ (numeric), $Binary\_label\_name$, $Binary\_label$ (Quartz vs. Non-Quartz), $Major - axis$ and $Minor - axis$ (grain size in mm), x_annot, y_annot (coordinates of the annotated point based on the point-counting method), x_inter, y_inter (intersection point of the annotated paths), and $path\_1$, $path\_2$ (HTML path elements). All coordinates and HTML paths are in reference to the original high-resolution mosaic image of the thin section. To convert these values to the 256×256 patch-level reference, one must center the coordinate system on (128, 128), corresponding to the transformed position of (x_inter, y_inter). The main folder also contains two partition files, $Fold1\_complete\_info\_allclasses.csv$ and $Fold2\_complete\_info\_allclasses.csv$, specifying the relative paths to the PPL-0.png image of each grain assigned to the respective fold. As described in the main text, all grains from a given thin section belong exclusively to either Fold 1 or 2. The code provided below illustrates the conversion of path HTML elements to vectors in the patch space.

```
from svgpathtools import parse_path
import pandas as pd

    annot_info = pd.read_csv(annot_file)
    path_1 = annot_info["path_1"][0]
    path_2 = annot_info["path_2"][0]
    data_path = {'path_1':path_1,'path_2':path_2}

    center_coords_or = [annot_info["x_inter"],annot_info["y_inter"]]
    center_coord_patch = [patch_size/2, patch_size/2]

    dx = center_coord_patch[0] - center_coords_or[0]
    dy = center_coord_patch[1] - center_coords_or[1]

    line_points = {'path_1':[],'path_2':[]}
    both_paths = ['path_1','path_2']

    image = Image.open(image_path)
    draw = ImageDraw.Draw(image)

    for path_name in both_paths:
        path_x = data_path[path_name]
        path_x = parse_path(path_x)
```

```
        end_point = ((path_x.start.real*24786)+dx,
                     (path_x.start.imag*24786)+dy)


        start_point = ((path_x.end.real*24786)+dx,
                     (path_x.end.imag*24786)+dy)

        line_points[path_name] = [start_point,end_point]

    draw.line(line_points['path_1'], fill=(255,0,0), width=10)
    draw.line(line_points['path_2'], fill=(255,0,0), width=10)
```

Listing 1: Convert HMTL paths to vectors.

2 Architecture Complexity and Inference Performance

Table 1: Model complexity and average inference time for binary (2-class) and multi-class (25-class) classification. Params in millions (M), FLOPs in billions (B).

Model complexity and inference performance				
DL Architecture	**Task**	**Params (Trainable / Total)**	**FLOPs (B)**	**Avg. Time (ms)**
ResNet	Binary	23M / 23M	4.14	4.46
GoogLeNet	Binary	5M / 5M	1.51	5.18
ViT	Binary	303M / 303M	59.7	17.45
Swin Transformer	Binary	86.9M / 86.9M	17.1	25.91
LITHOS Baseline	Binary	67M / 673M	133.26	40.18
ResNet	Multi-class	23M / 23M	4.14	4.46
GoogLeNet	Multi-class	5M / 5M	1.51	5.18
ViT	Multi-class	303M / 303M	59.7	17.60
Swin Transformer	Multi-class	86.9M / 86.9M	17.1	26.07
LITHOS Baseline	Multi-class	67M / 673M	133.28	40.18

Table 1 shows the model complexity and inference performance for all architectures evaluated in both binary and multi-class classification tasks. The table compares the number of trainable and total parameters, the computational cost in FLOPs, and the average inference time per image. The LITHOS Baseline has a large total parameter count and a high number of FLOPs compared to the other baselines, due to its dual ViT encoders. However, these encoders are kept frozen during training, meaning only about 10% of the parameters are updated. This design reduces the effective training cost and memory consumption while leveraging strong, pretrained features from each polarization stream, which is key to improving classification performance.

3 Binary Precision-Recall Curves

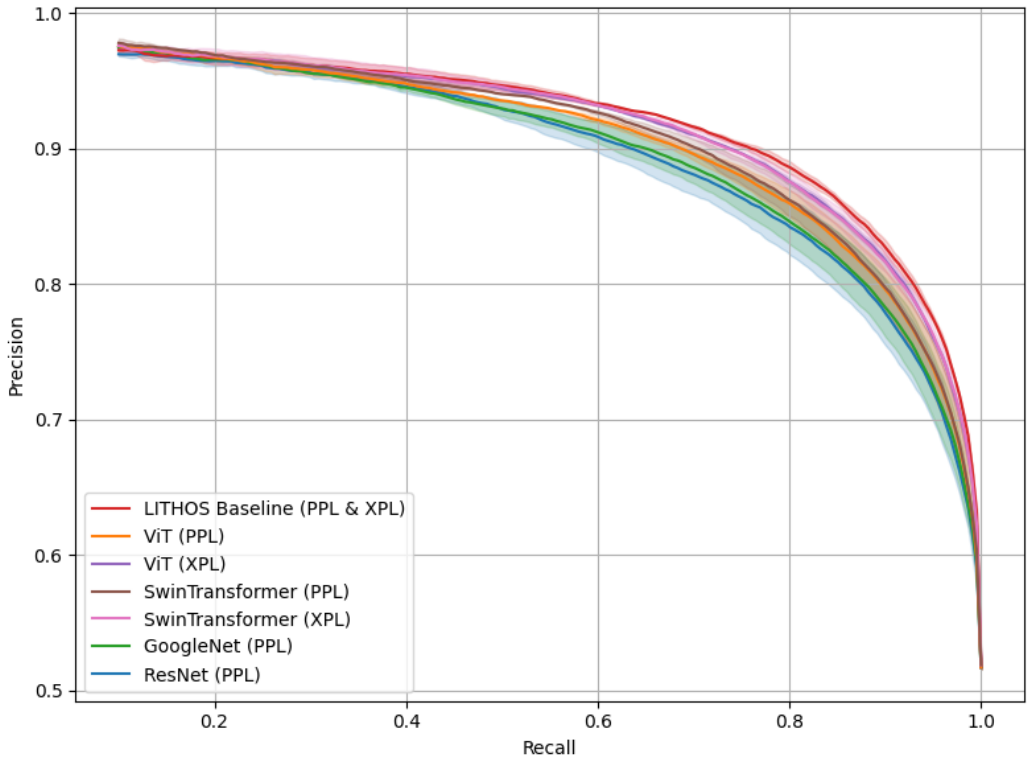
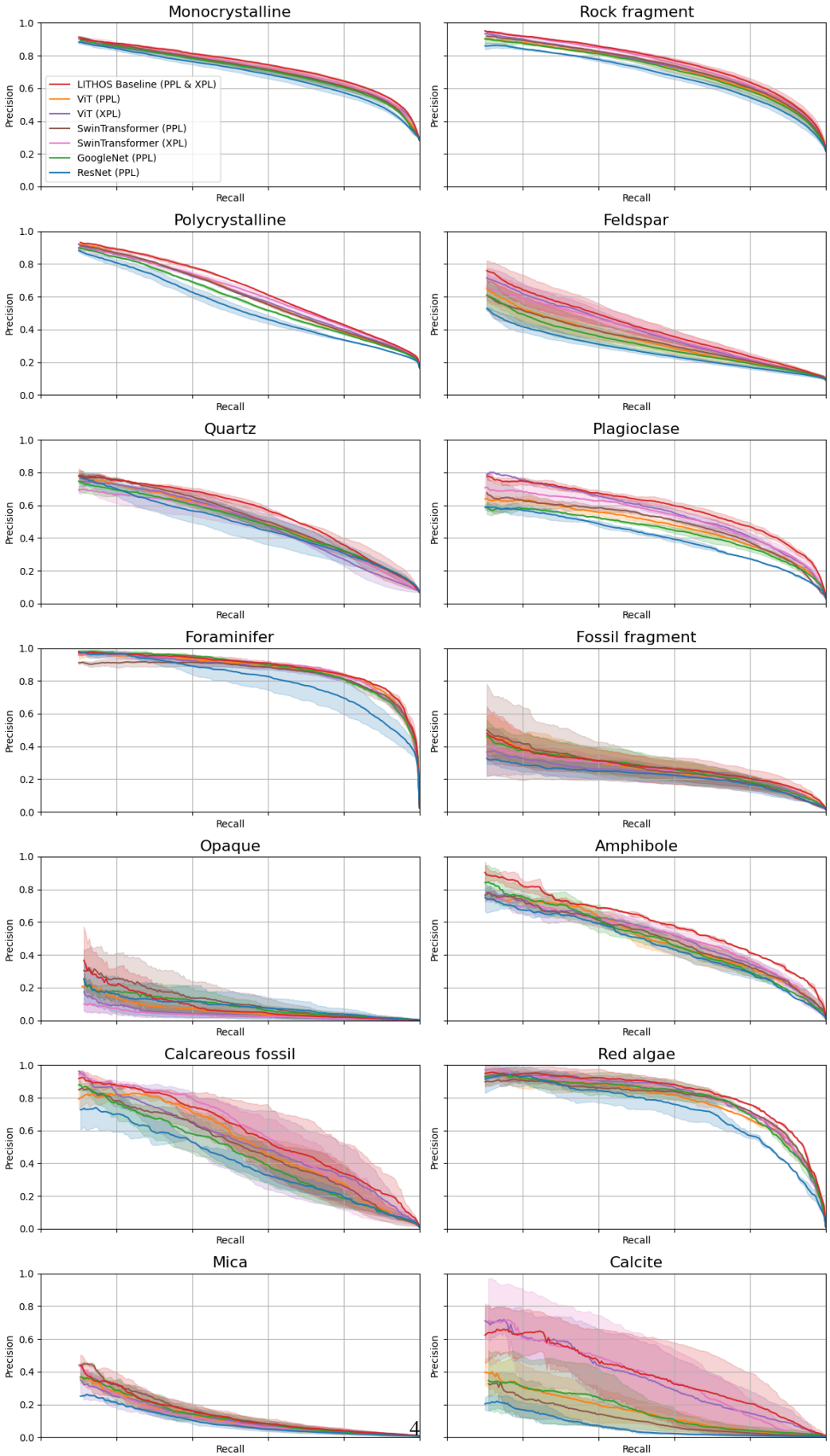


Figure 1: Precision-Recall curves across both folds (mean $\pm$ std) for the binary Quartz vs. No-Quartz classification task. Our LITHOS Baseline achive the best performance in both folds.

4 Multiclass Precision-Recall Curves


Monocrystalline
Rock fragment
Polycrystalline
Feldspar
Quartz
Plagioclase
Foraminifer
Fossil fragment
Opaque
Amphibole
Calcareous fossil
Red algae
Mica
Calcite
LITHOS Baseline (PPL & XPL)
ViT (PPL)
ViT (XPL)
SwinTransformer (PPL)
SwinTransformer (XPL)
GoogleNet (PPL)
ResNet (PPL)
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0

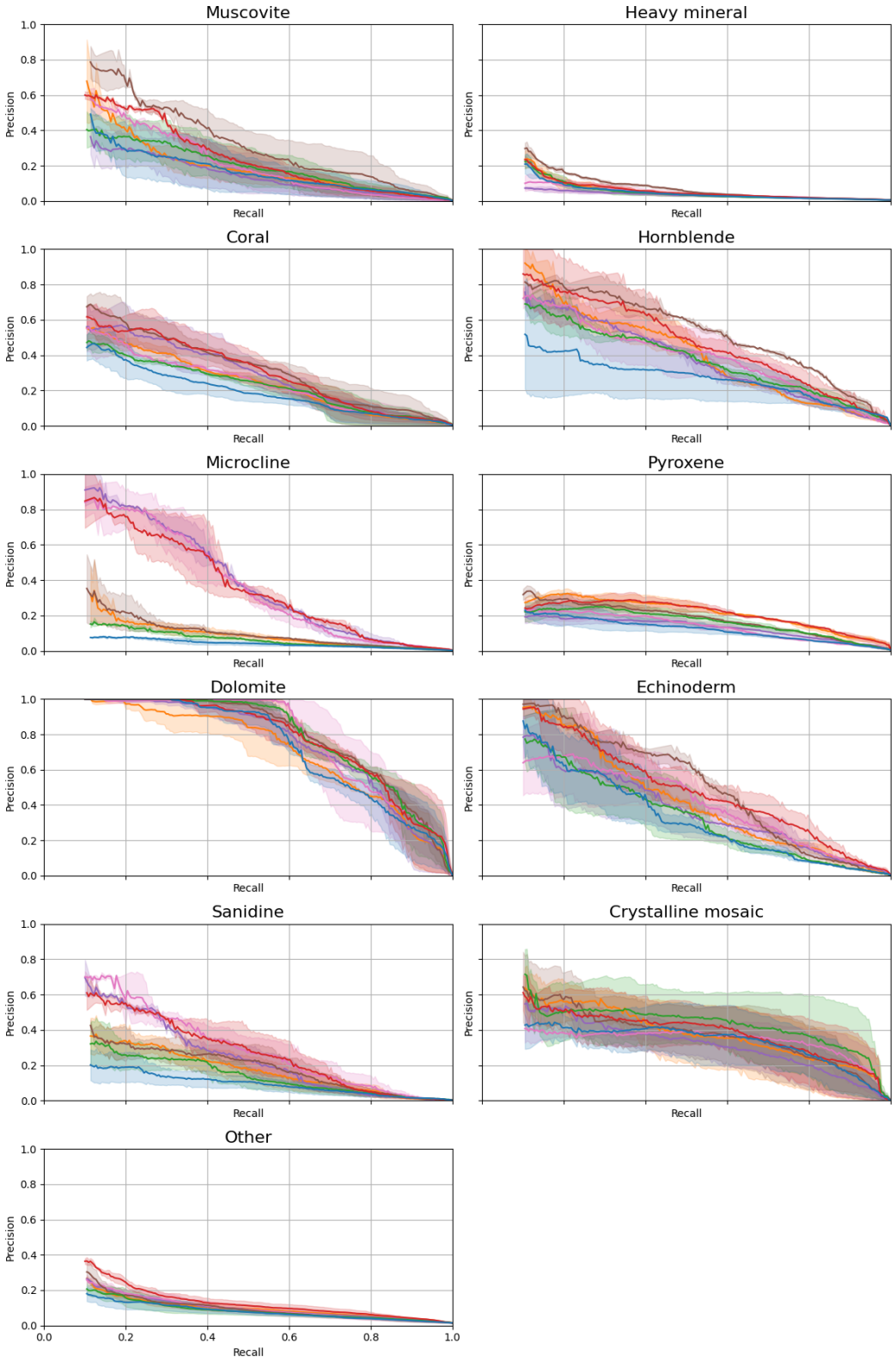


Figure 2: Precision-Recall curves across both folds (mean $\pm$ std) for the multiclass classification task for the 25 minerals in the LITHOS Dataset. Our LITHOS baseline achieves higher performance in the most represented mineral classes of our database. In some classes, there is a noticeable increase in performance on models that incorporate XPL images of the minerals, instead of PPL-only models. This behavior might be related to distinctive features that are revealed under cross-polarized light. For example, cross-hatched twinning patterns reveal microcline's gridiron structure hidden in featureless PPL views.

5 Multiclass Confusion Matrix

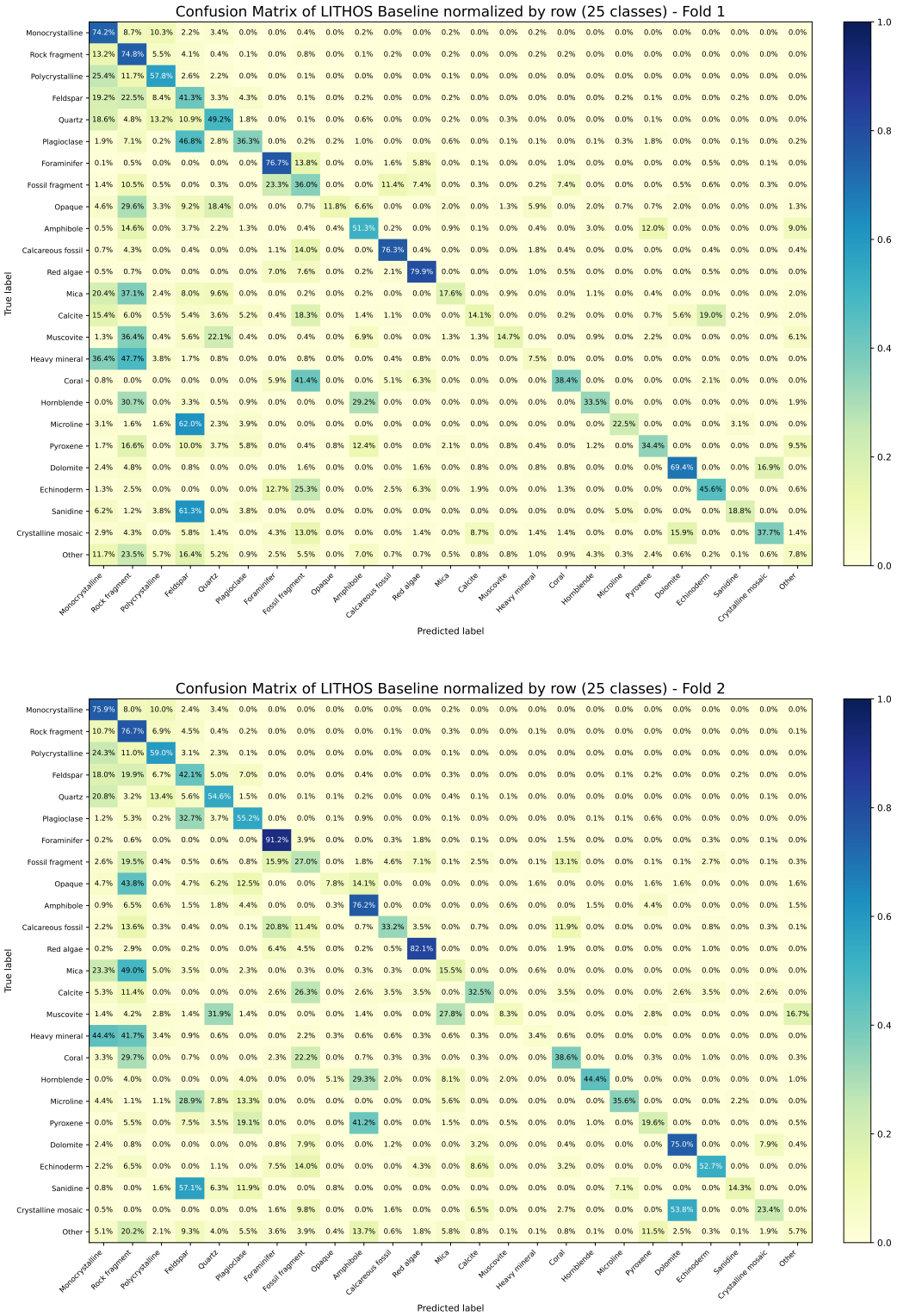


Figure 3: Confusion matrices of the LITHOS Baseline model for the 25-class mineral classification task across both data folds.