Model Comparison on Fetal Head Abnormalities Dataset

Meripe Sathvik
220010031

January 23, 2025

Contents

1 Introduction

1.1 Problem statement

The objective of this assignment is to analyze and compare the performance of state-of-the-art deep learning models for the classification of fetal head abnormalities using the Fetal Head Abnormalities Dataset available on Kaggle.

1.2 Objective

1. Implement the ResNet50 model.
2. Implement the MobileNet v3-small model.
3. Using traditional augmentation techniques.
4. Using advanced augmentation techniques.
5. Using Few shot learning to train the models.

1.3 Models

The models chosen were ResNet50 and MobileNet v3-small. I used these pre-trained models and fine tune them to solve a classification model on the dataset Fetal Head Abnormalities. I used various metrics like Accuracy, Precision, Recall, F1-Score and Confusion matrix to understand how they are performing.

2 Methodology

2.1 Dataset

The dataset contains 14 classes. Total samples are 974, 278 and 139 in Train set, Validation set and Testing set respectively. The image and their respective classes are mentioned in `_classes.csv` in each of the train, valid, test folder.

2.2 Data Preprocessing

2.2.1 Augmentation

First model was based on standard augmentation techniques such as rotation, flipping and was fine tuned. For second model I used advanced augmentation technique like cutmix.

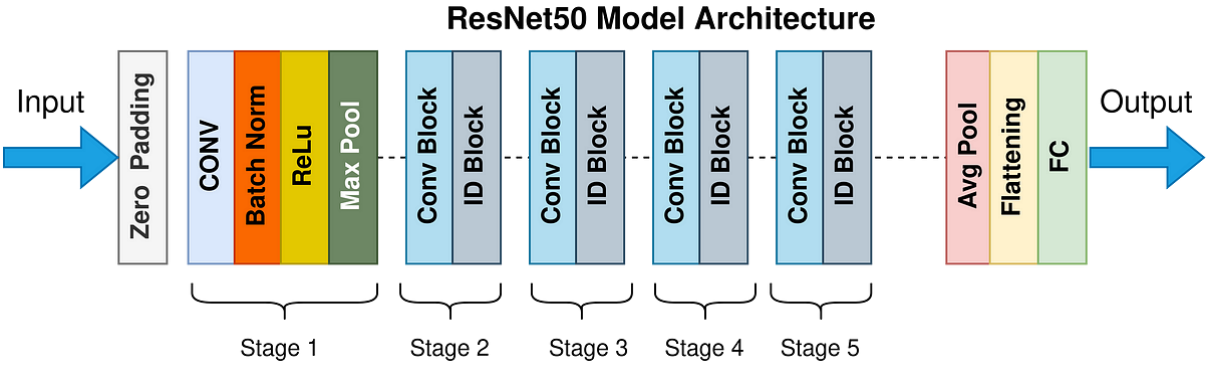
2.2.2 Few Shot Learning

Still implementing few shot learning.

2.3 Model Architectures

2.3.1 ResNet50

ResNet50 is a deep convolutional neural network with 50 layers. It introduces "residual connections" (or skip connections), which help mitigate the vanishing gradient problem by allowing gradients to flow more effectively through the network.

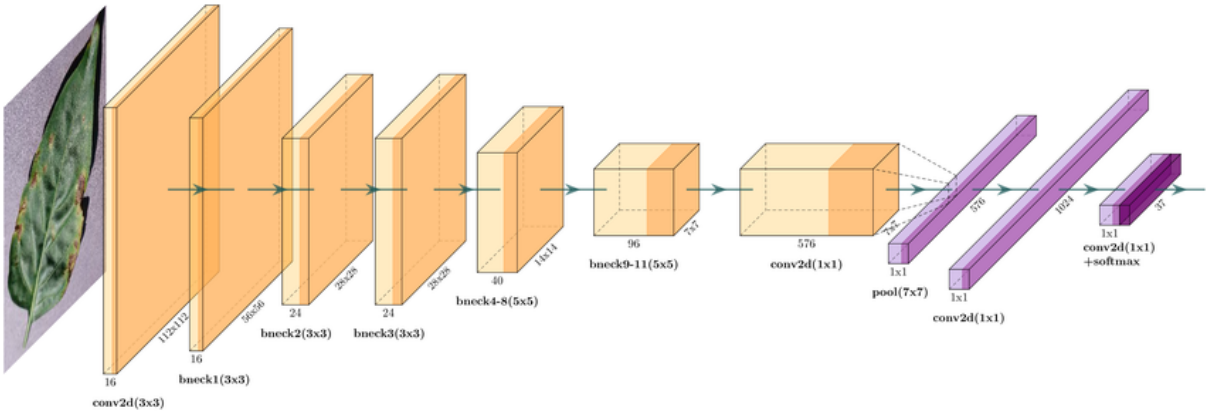


I have confirmed the layers and about FC with the code. I matched the output of FC to number of classes in our dataset and trained the FC layer.

```
for name, param in model.named_parameters():
    param.requires_grad = 'fc' in name
model.fc = nn.Linear(model.fc.in_features, num_classes)
```

2.3.2 MobileNet v3-small

MobileNet v3-small is a lightweight convolutional neural network designed for resource-constrained environments, such as mobile and edge devices. It utilizes efficient building blocks like depthwise separable convolutions and squeeze-and-excitation layers.



While building the model, here also I confirmed the layers with the code and matched the output of the FC layer with number of classes in our dataset.

```
for param in model.parameters():
    param.requires_grad = False
for param in model.classifier.parameters():
    param.requires_grad = True
model.classifier[-1] = nn.Linear(model.classifier[-1].in_features, num_classes)
```

2.4 Training Setup

2.4.1 Hyperparameters

List and explain the hyperparameters used (learning rate, batch size, epochs, etc.). For all the models, I have followed these hyperparameters `batch_size=32, img_size=224, epochs=50,lr=0.0001`

2.4.2 Training Process

1. **Loss Function**:
    - The standard `CrossEntropyLoss` function from PyTorch was used for training.
2. **Optimizer**:
    - The Adam optimizer was used for updating model parameter, i.e. for adaptive learning rate adjustments.
3. **Training Metrics**:
    - For each epoch, I tracked the following metrics : Training loss, Validation loss and Validation accuracy.
4. **Training Workflow**:
    - Each epoch consisted of a forward pass to compute predictions, a loss computation step, and a backpropagation step to update weights.
    - For validation, the model was evaluated on unseen data without updating its parameters.

2.4.3 Implementation Details

1. **Libraries and Frameworks Used:**
    - The project utilized the following libraries and modules : torch, torchvision, numpy, pandas, scikit-learn.
2. **Execution Environment:**
    - I did whole project on HPC for accelerated training.
3. **Data Handling:**
    - The dataset was preprocessed using the `ImageDataset` class, and batched using the `get_data_loaders` function from `preprocessing_data.py`.
    - During dataset loading, statistics such as the total number of samples and class distribution were printed for verification.
4. **Model Implementation and Training:**
    - The models were implemented in two separate files:
        - `train_resnet50.py` for building and training the ResNet50 model.

  - `train_mobilenet_v3.py` for building and training the MobileNet v3-small model.
- The use of hyperparameters such as learning rate, batch size, and number of epochs, were defined in these files and could be adjusted for experimentation.
- After each epoch, the validation loss and validation accuracy were computed and printed using the `evaluate_model()` function.

5. **Metrics and Visualization:**
   - The training and evaluation metrics (e.g., accuracy, precision, recall, F1-score, and confusion matrices) were saved in `.json` format for each model.
   - These metrics were later analyzed and visualized using `generate_visualizations.py`.

3 Experimental Results

3.1 Evaluation Metrics

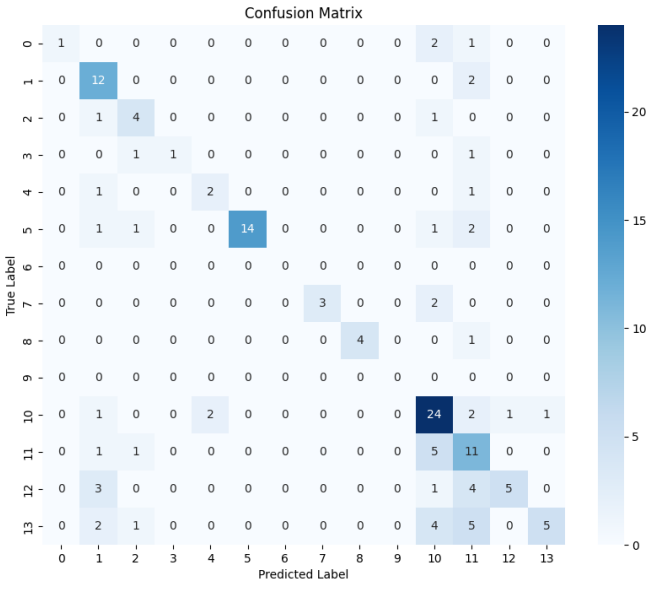
The metrics used were Accuracy, F1-Score, Precision, Recall, Confusion Matrix.

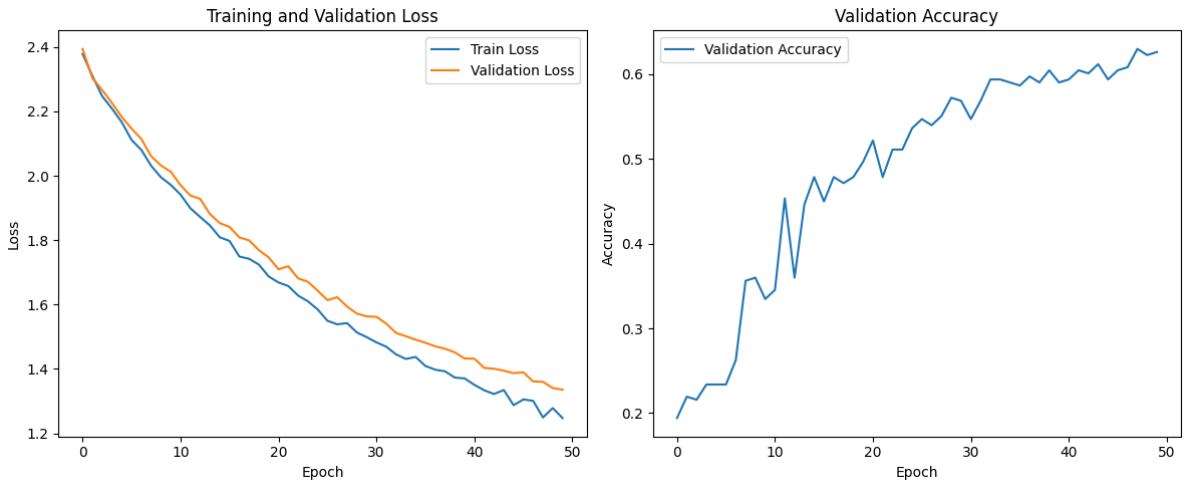
```
cm = confusion_matrix(all_labels,all_preds,
        labels=np.arange(num_classes))
'accuracy': np.mean(all_preds == all_labels)
precision_score(all_labels, all_preds, average='weighted', zero_division=0)
recall_score(all_labels, all_preds, average='weighted', zero_division=0)
f1_score(all_labels, all_preds, average='weighted', zero_division=0)
```

3.2 Results for ResNet50

3.2.1 Standard Augmentation

TEST metrics were:
`Accuracy: 0.6187`
`Precision: 0.7111`
`Recall: 0.6187`
`F1-Score: 0.6160`

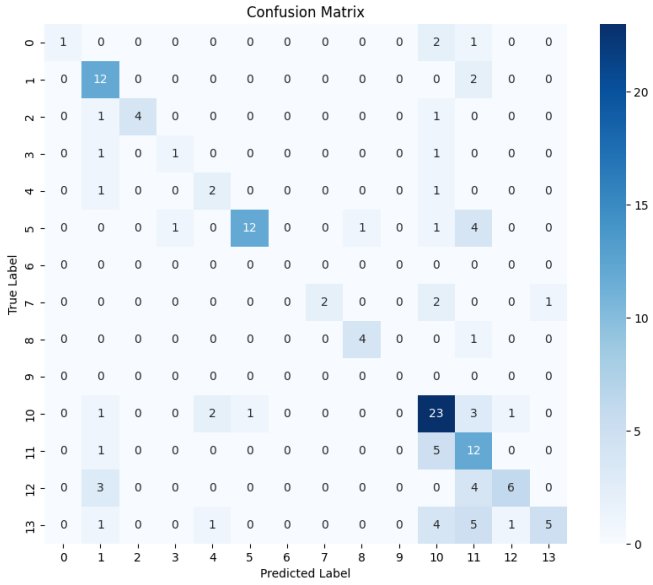


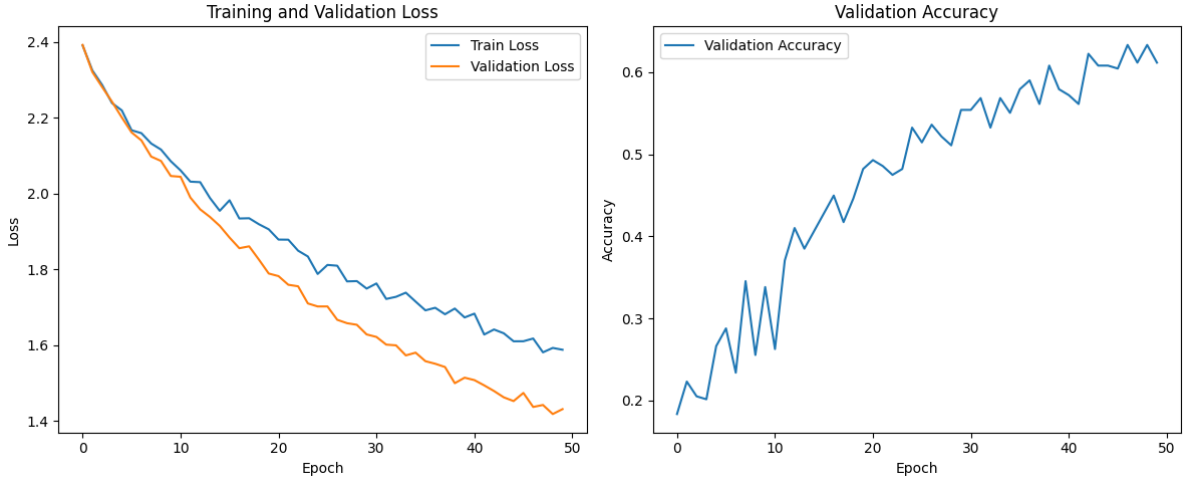


3.2.2 Advanced Augmentation

TEST metrics were:

```
Accuracy:  0.6043
Precision:  0.6916
Recall:  0.6043
F1-Score:  0.6016
```

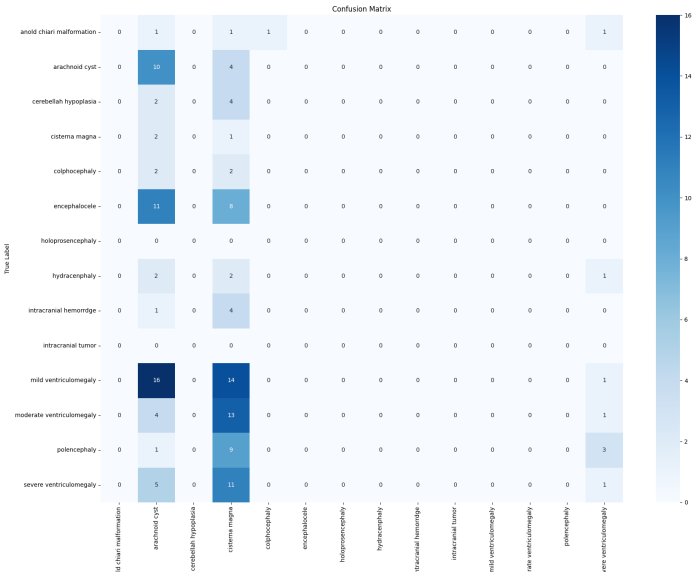




3.2.3 Few Shot Learning

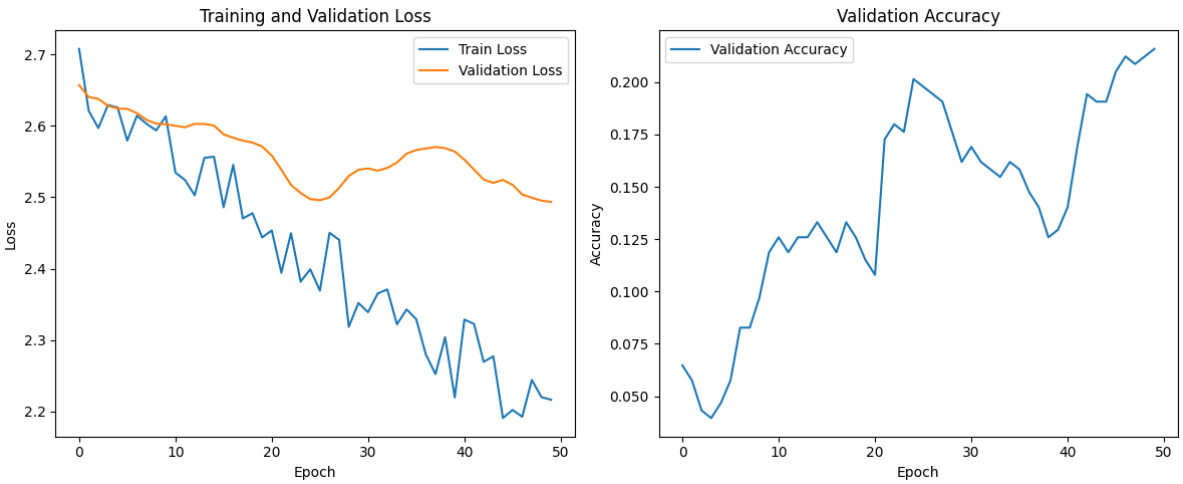
Zero shot learning - inference

TEST metrics were:
`Accuracy: 0.0863`
`Precision: 0.0333`
`Recall: 0.0072`
`F1-Score: 0.0387`



Few shot learning

TEST metrics were:
`Accuracy: 0.2158`
`Precision: 0.3694`
`Recall: 0.2158`
`F1-Score: 0.2090`

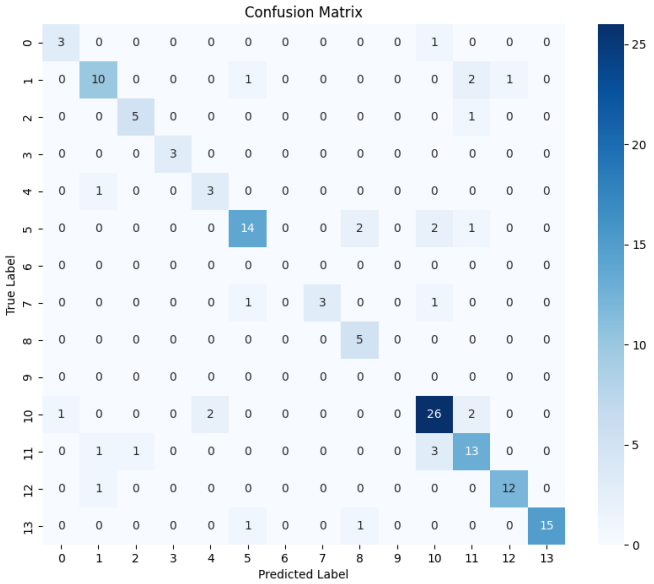


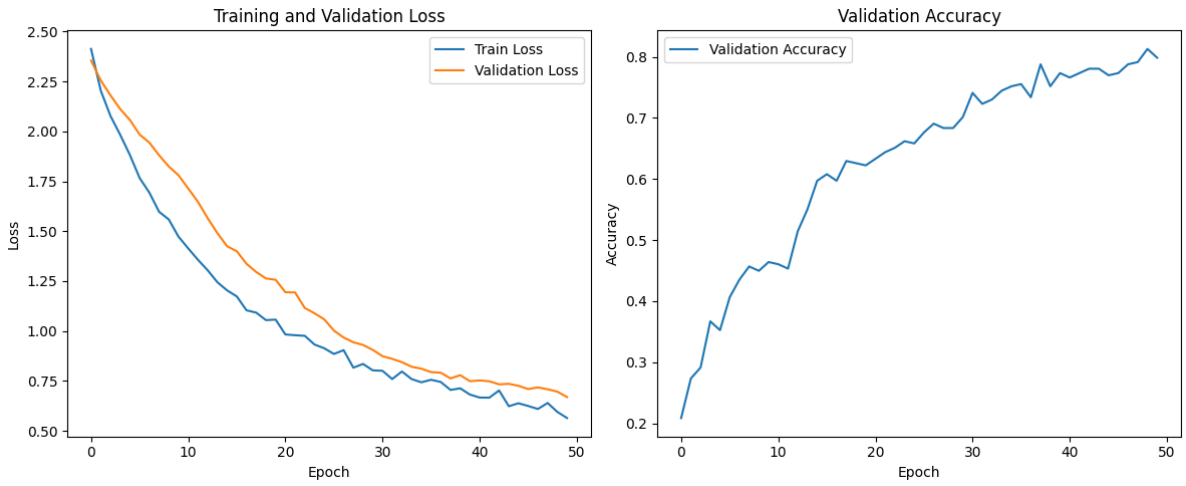
3.3 Results for MobileNet v3-small

3.3.1 Standard Augmentation

TEST metrics were:

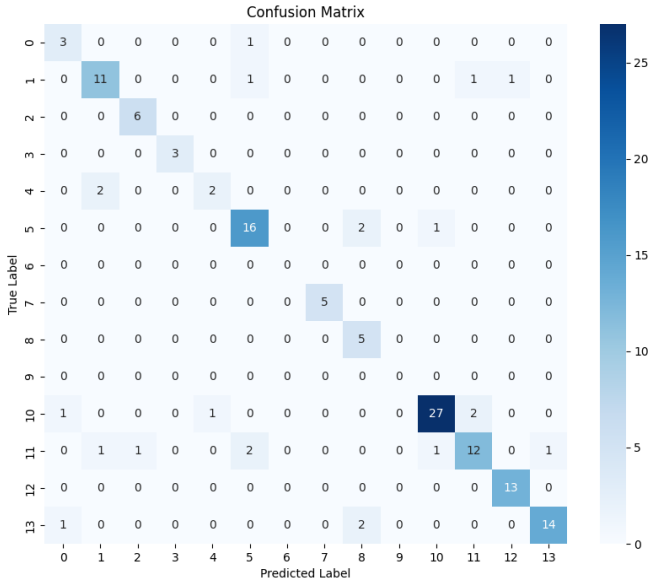
`Accuracy: 0.8058`
`Precision: 0.8179`
`Recall: 0.8058`
`F1-Score: 0.8071`

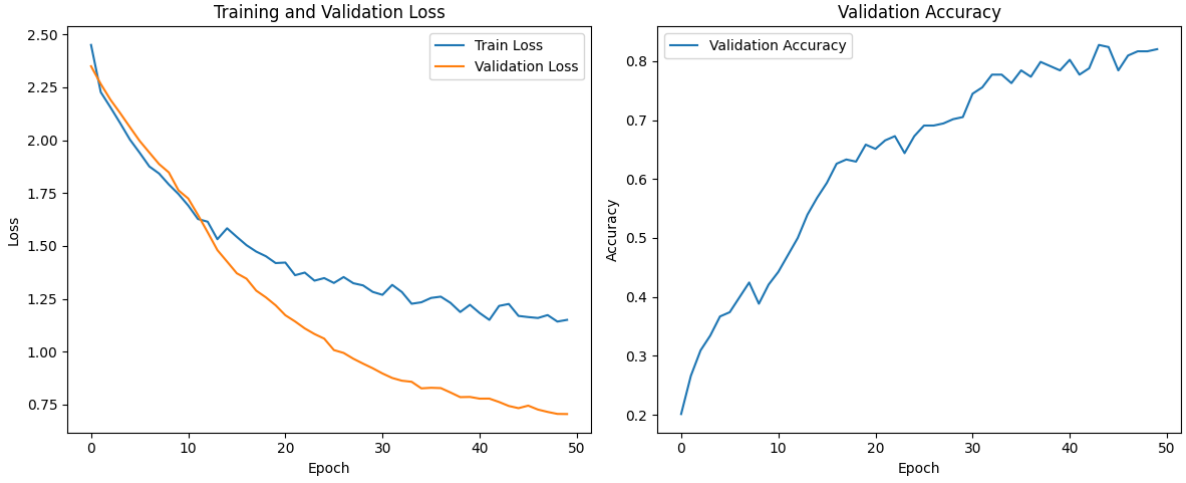




3.3.2 Advanced Augmentation

TEST metrics were:
`Accuracy: 0.8417`
`Precision: 0.8517`
`Recall: 0.8417`
`F1-Score: 0.8420`

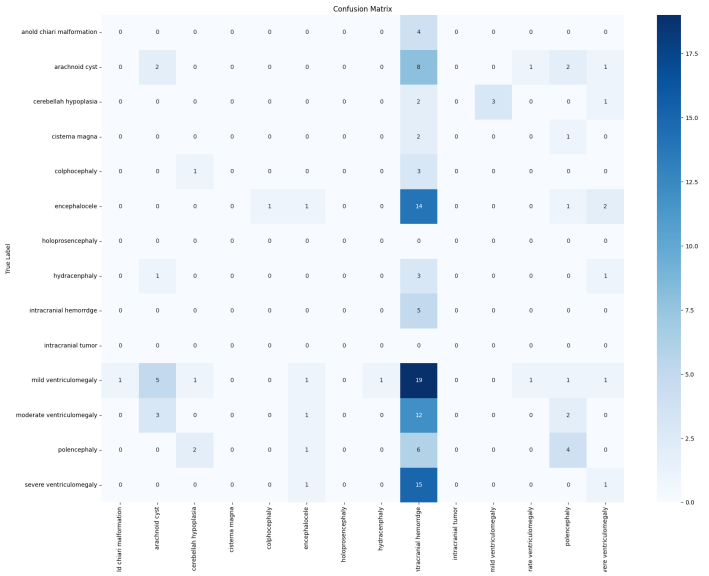




3.3.3 Few Shot Learning

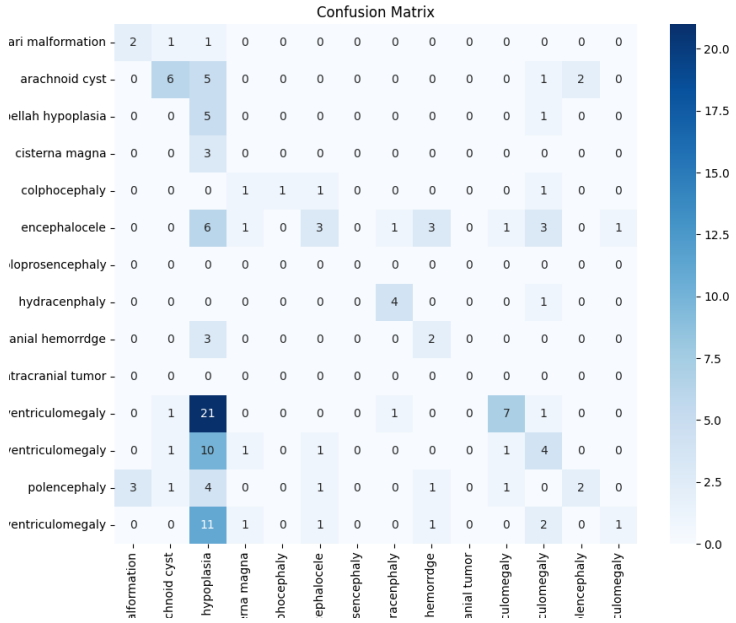
Zero shot learning - inference

TEST metrics were:
`Accuracy: 0.0935`
`Precision: 0.0991`
`Recall: 0.0725`
`F1-Score: 0.0664`

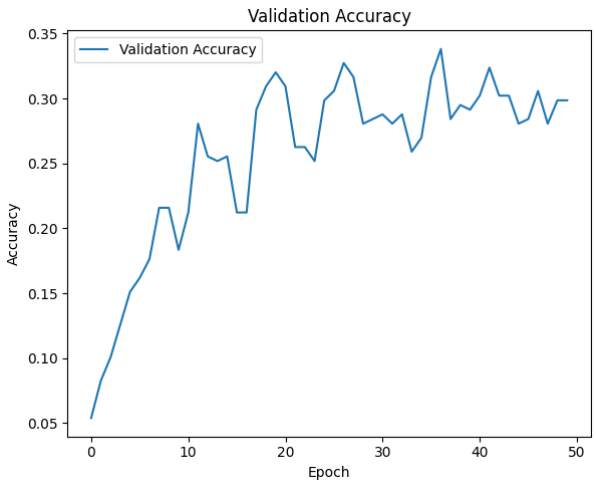


Few shot learning

TEST metrics were:
`Accuracy: 0.2662`
`Precision: 0.4977`
`Recall: 0.2662`
`F1-Score: 0.2935`




Training and Validation Loss
Train Loss
Validation Loss
Loss
Epoch
3.0
2.5
2.0
1.5
1.0
0.5
0
10
20
30
40
50


Validation Accuracy
Validation Accuracy
Accuracy
Epoch
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
10
20
30
40
50