# Summary

- "Last layers retrain" method (=fine-tuning) – a technique used to adapt a pre-trained model to a specific task without the need to retrain all of its layers from scratch.
  Originally trained on the ImageNet dataset, ResNet50 extracts features like edges and patterns in its early layers. During pre-training, these early layers are frozen, meaning their weights remain unchanged throughout training epochs. Meanwhile, the weights of the final layers are updated to learn task-specific patterns, aiming to reduce computational load and mitigate the risk of overfitting.

- **<u>My code steps:</u>**
  1. **Dataset**:
     - I used the following division:
       - **Training:** 10 SVS files with corresponding labels in *'train.csv'*. 70% of samples are negatives, 30% are positives.
       - **Validation**: 2 SVS files with corresponding labels in *'valid.csv'.* 50% positives and 50% negatives.
       - **Test**: 2 SVS files with corresponding labels in *'test.csv'*. 50% positives and 50% negatives.
     - **'DPDataset' class** is a custom dataset. Its getitem method processes SVS files using OpenSlide package as follows:
       - Iterates through slide levels, dividing each into 512x512 tiles.
       - The tiles are converted to 'RGB' format and are rescaled to [0, 1] to maintain consistency across different devices.
       - Each level represents a different pyramid image of the slide with a different resolution, where the first layers represent a higher resolution image which captures more details, while the last layers provide global context.
       - To exclude background tiles, they undergo a validation process where the proportion of white pixels in each tile is calculated. A threshold of 0.9 is set, so only tiles with a lower proportion of white pixels are considered valid. Additionally, a condition is applied that the tile's standard deviation must be higher than 0.04, as background images typically have more uniform pixel values.
       - Valid tiles are resized to 256x256 (ResNet50 input size) and normalized using ImageNet mean and standard deviation values:
         (mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]).
       - The getitem methods returns a tensor with the following shape:
         (#tiles per SVS file, #channels, Height, Width).

  2. **Model:** The 'CustomResNet50' class loads a pre-trained ResNet50 model with all weights frozen except for the last layer. A fully connected layer is added to adjust the model for the binary classification of positive and negative classes.
     During the forward method, all valid tiles of the SVS file are passed through the model to extract outputs. These outputs are processed through a sigmoid function to obtain probabilities, and the slide-level prediction is calculated by averaging all tile predictions.

  3. **Loss:** A weighted binary cross entropy loss (0.3 for the negative, 0.7 for positive) to address class imbalance.

4. **Optimizer:** Adam was used as an optimizer.
5. A config file named hparams.json is used to define the hyperparameters.
6. main.py script is used for training the model.

**Model Evaluation:**

** Since the data was tagged manually, the results lack reliability. However, here are the evaluation methods I would use with a real dataset:

During training I evaluated the model performance by loss vs. epoch and accuracy vs. epoch graphs, as shown in figure (2). Due to computational power limitations, only 10 epochs were used.

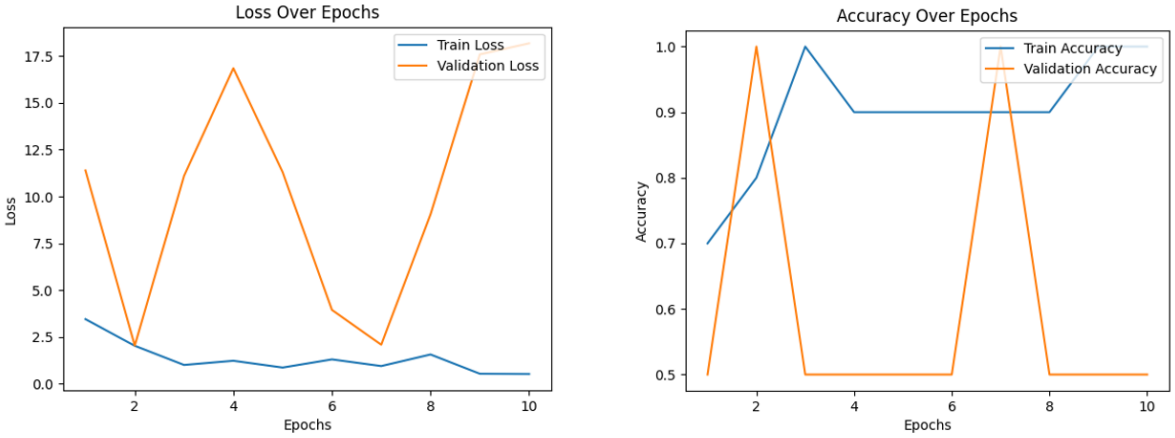


Figure (1) – Train & validation loss and accuracy

The evaluation on the test set includes the following methods:

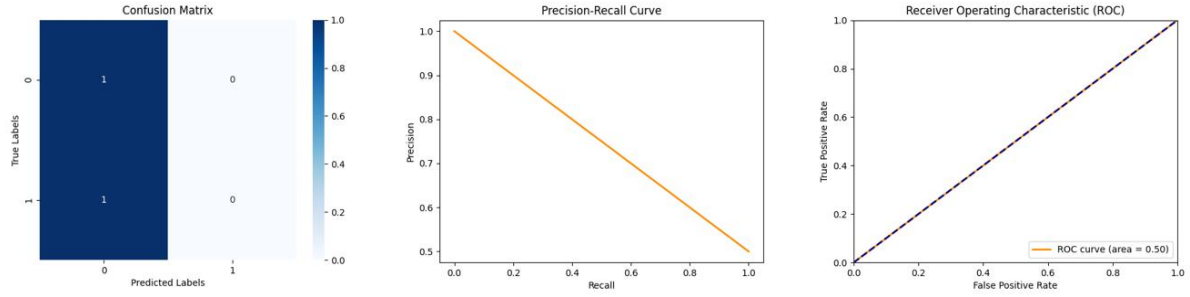


Figure (2) – Confusion matrix, PR-curve, ROC-curve evaluation on the test set

```
Test Accuracy: 0.5000
Test F1 Score: 0.0000
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

**Future Steps:**

- Save only the valid tiles as PNG files to reduce training run time.

- **Advanced preprocessing:**
    - Overlapping Tiles: Consider taking overlapping tiles to capture context around the edges.
    - Correlation Threshold: Calculate the correlation between subsequent tiles and set a threshold to remove highly correlated tiles, ensuring no important clinical information is lost.
    - Pen Marks Removal: Remove pen marks using color thresholding and inpainting techniques.
    - Contrast Enhancement: Apply histogram equalization or CLAHE.
    - Texture-Based Filtering: Implement stricter criteria for tile validity based on the texture that defines the pathology.

- **Data augmentations:** Apply augmentations such as elastic deformation, rotation, flipping, scaling, and color jittering to increase the dataset size and improve model robustness. Note: Augmentation was not previously performed as it didn't influence the randomly tagged data.

- **Models**:
    - Use different CNNs, ViTs architectures for the classification model.
    - Multi-tile score calculation – test different approaches, such as encoding each tile and then aggregating the embedded vectors for prediction.

- Implement cross-validation to ensure model reliability and performance.
- Consider using the SlideFlow package:
  https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-024-05758-x
  https://github.com/jamesdolezal/slideflow