1 Top-level summary

1.1 Process

1. Starting point: copied the model (ResNet-18) and training hyperparameters from the research paper
2. Split training data into training (80%) and validation (20%), stratified on annotator count. Used holdout validation instead of cross-validation for training speed.
3. Used early stopping on the training process triggered on validation F1 score; loaded the model state from the best epoch
4. ResNet-18 experiments: image augmentation, removing difficult images, class balancing, and transfer learning
5. Using the best hyperparameters from the ResNet-18 experiments, I moved on to ViT-Base Patch 16/224 model
6. ViT experiments: learning rate, effective batch size (gradient accumulation), dropout, and weight decay
7. Tried ensembling with the best ResNet and ViT models
8. Using the best hyperparameters from the ResNet-18 experiments, trained and tested FCNN with 4 fully-connected layers (128 neurons → 64 neurons → 32 neurons → 1 neuron)

1.2 Performance

1. ResNet-18 (11 million parameters) – comparable performance to the research paper (paper reported AUC); better performance than the research paper with additional generalization techniques (however, made a mistake in the training process – see Mistakes/issues section). Only trained and tested once; research paper trained and tested for 10 random seeds.
2. ViT-Base Patch 16/224 (86 million parameters) – better performance than research paper over 10 random seeds; unclear if performance is better than ResNet.
3. Ensemble ResNet+ViT – worse performance than each model individually.
4. FCNN (19 million parameters) – the worst model but better than randomly guessing

Model	AUC
ResNet-18, random init weights (research paper)	84.5
Research paper, ResNet-18, pretrained (research paper)	92.7
ResNet-18, random init weights	91.6
ResNet-18, pretrained	94.2
ViT-Base Patch 16/224, pretrained	94.85

FCNN	74.3

1.2.1 Research paper comparison

I tried to replicate the training configuration from the paper. Performance is comparable.

Training hyperparameters:

1. Single training/testing run
2. Randomly initialized weights and pretrained weights; update all layers
3. Holdout validation split 80/20
4. Image augmentation on all samples (horizontal flip p=0.5, vertical flip p=0.5, rotate +/- 180 degrees p=0.5)
5. Differences from the research paper
    a. I used holdout validation, but the research paper used the test set for validation and for reporting performance
    b. I only trained and tested once; the research paper trained and validated/tested on 10 random seeds

Research paper performance:

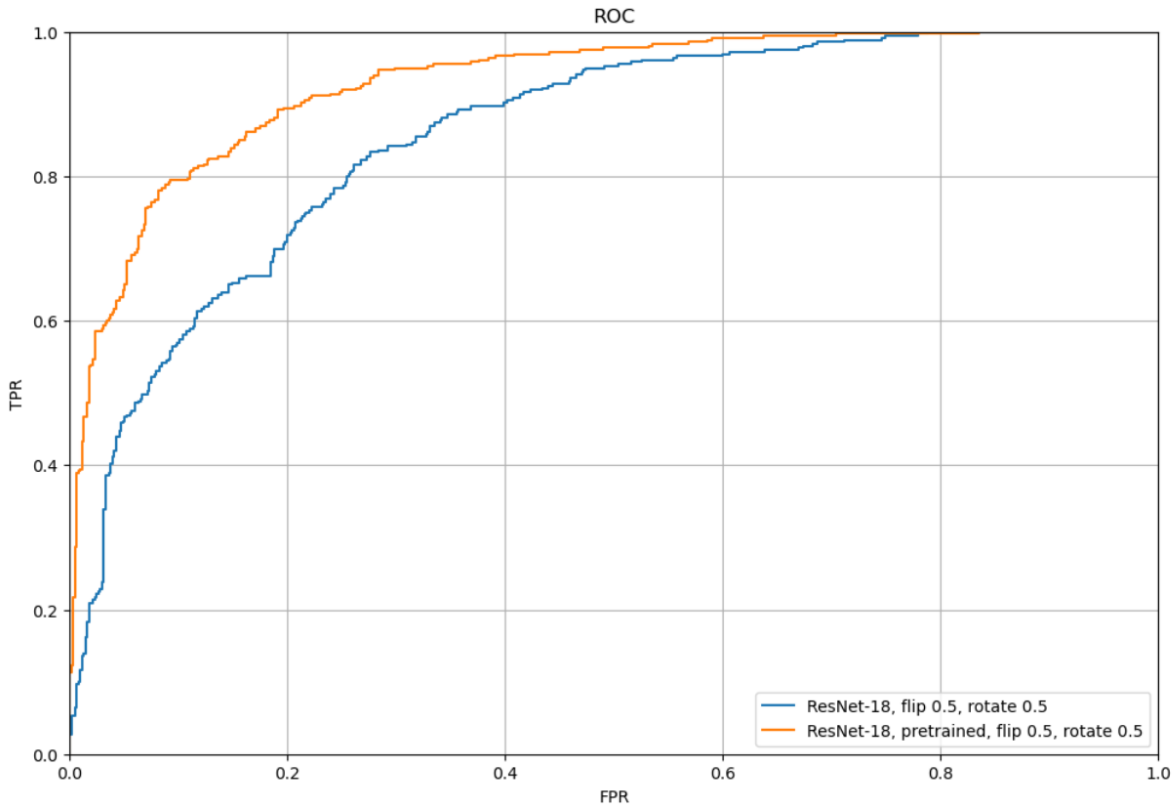
Pretraining?	AUC (%) on test set by training set size $n = 100$	$n = 200$	$n = 400$	Full
No	67.4 ± 3.1	73.6 ± 3.7	79.3 ± 2.3	84.5 ± 1.1
Yes	$\mathbf{83.7 \pm 1.7}$	$\mathbf{89.3 \pm 1.8}$	$\mathbf{92.4 \pm 0.7}$	$\mathbf{92.7 \pm 0.4}$

Table 3: Using weights pretrained on ImageNet significantly improves the performance of ResNet-18 on our dataset. n indicates the number of images per class used for training. Means and standard deviations shown are for 10 random seeds.

For more-robust evaluation, for each model we consider the five highest AUCs on the test set, which are evaluated at every epoch. We report the mean and standard deviation of these values calculated over 10 different random seeds.

My performance:

Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
ResNet-18, flip 0.5, rotate 0.5	15.091	0.721	0.639	0.856	0.677	0.776	0.747	0.855
ResNet-18, pretrained, flip 0.5, rotate 0.5	12.515	0.835	0.789	0.909	0.811	0.865	0.849	0.929



1.2.2 Best of ResNet-18

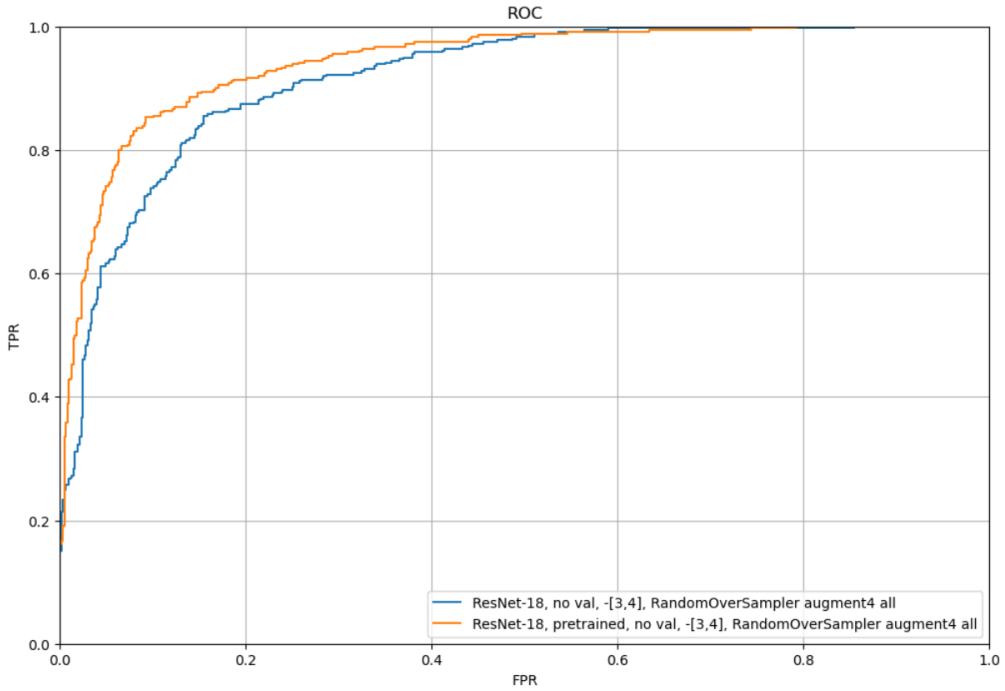
Training hyperparameters:

1. Single training/testing run
2. Randomly initialized weights and pretrained weights; update all layers
3. No holdout validation
4. Remove 3/7 and 4/7 annotator images from training set*
5. Random oversampling*
6. Image augmentation on all samples

*These are additional generalization techniques I added on top of the research paper.

I did not try a deeper ResNet because ResNet-18 is already overfitting, and the research paper also found that deeper ResNets don't improve performance.

Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
ResNet-18, no val, -[3,4], RandomOverSampler augment4 all	12.995	0.795	0.764	0.885	0.779	0.840	0.824	0.916
ResNet-18, pretrained, no val, -[3,4], RandomOverSampler augment4 all	13.717	0.861	0.825	0.922	0.843	0.886	0.874	0.942



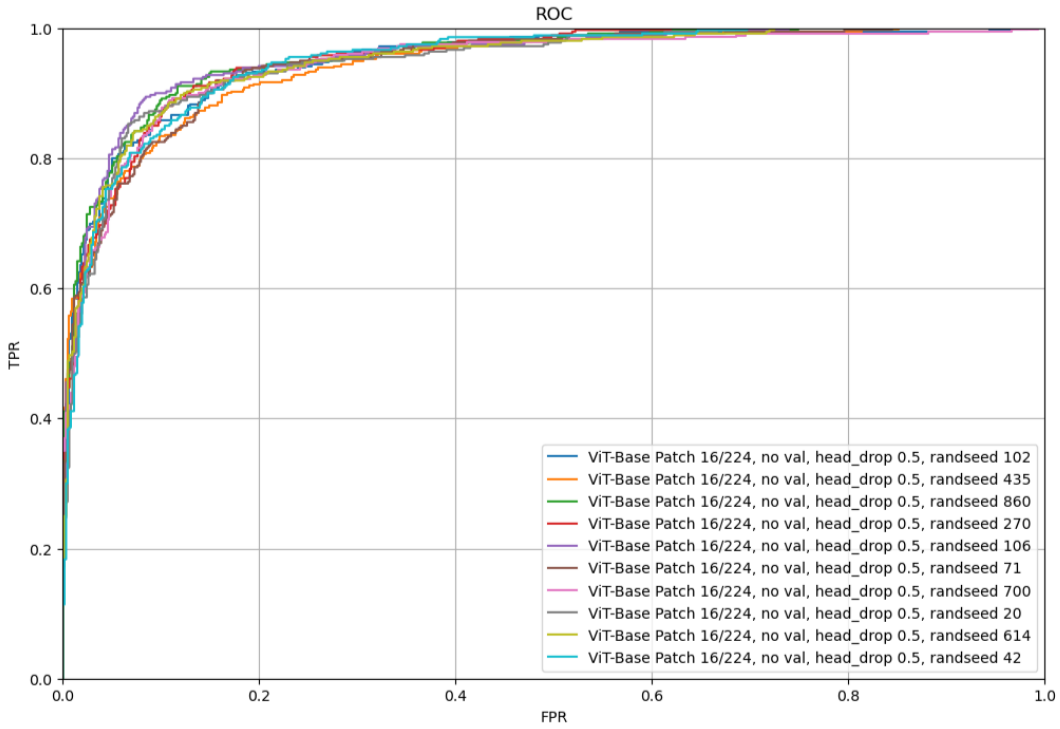
1.2.3 ViT-Base Patch 16/224

Training hyperparameters:

1. 10 training and testing runs over 10 random seeds
2. Pretrained weights; update all layers
3. No holdout validation
4. Dropout rate for final layer = 0.5 (0 for all others)
5. Same as ResNet-18:
    a. Remove 3/7 and 4/7 annotator images from training set
    b. Random oversampling
    c. Image augmentation on all samples

AUC mean and standard deviation: 0.9485, 0.0041

Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
ViT-Base Patch 16/224, no val, head_drop 0.5, randseed 102	79.196	0.853	0.836	0.916	0.844	0.886	0.876	0.950
ViT-Base Patch 16/224, no val, head_drop 0.5, randseed 435	64.354	0.838	0.819	0.908	0.829	0.875	0.864	0.942
ViT-Base Patch 16/224, no val, head_drop 0.5, randseed 860	75.411	0.895	0.803	0.945	0.846	0.893	0.874	0.955
ViT-Base Patch 16/224, no val, head_drop 0.5, randseed 270	68.769	0.848	0.850	0.911	0.849	0.888	0.880	0.951
ViT-Base Patch 16/224, no val, head_drop 0.5, randseed 106	65.914	0.894	0.817	0.943	0.853	0.897	0.880	0.954
ViT-Base Patch 16/224, no val, head_drop 0.5, randseed 71	83.680	0.803	0.847	0.878	0.824	0.867	0.863	0.945
ViT-Base Patch 16/224, no val, head_drop 0.5, randseed 700	87.019	0.899	0.694	0.955	0.784	0.859	0.825	0.945
ViT-Base Patch 16/224, no val, head_drop 0.5, randseed 20	79.782	0.889	0.825	0.940	0.856	0.898	0.883	0.947
ViT-Base Patch 16/224, no val, head_drop 0.5, randseed 614	71.787	0.889	0.778	0.943	0.830	0.882	0.861	0.948
ViT-Base Patch 16/224, no val, head_drop 0.5, randseed 42	73.056	0.832	0.842	0.901	0.837	0.879	0.871	0.948

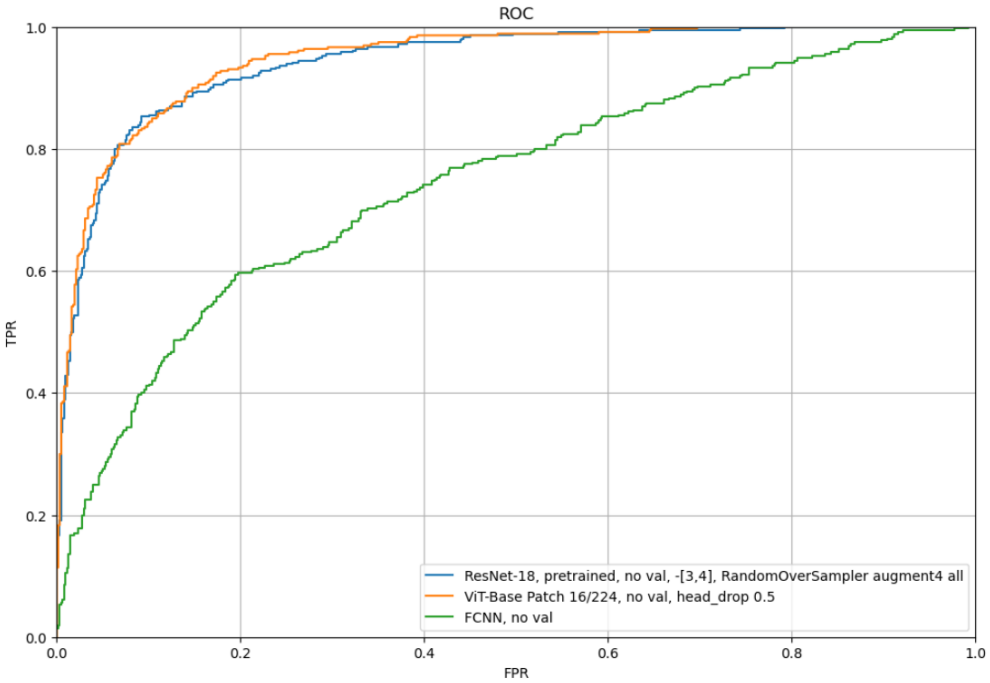


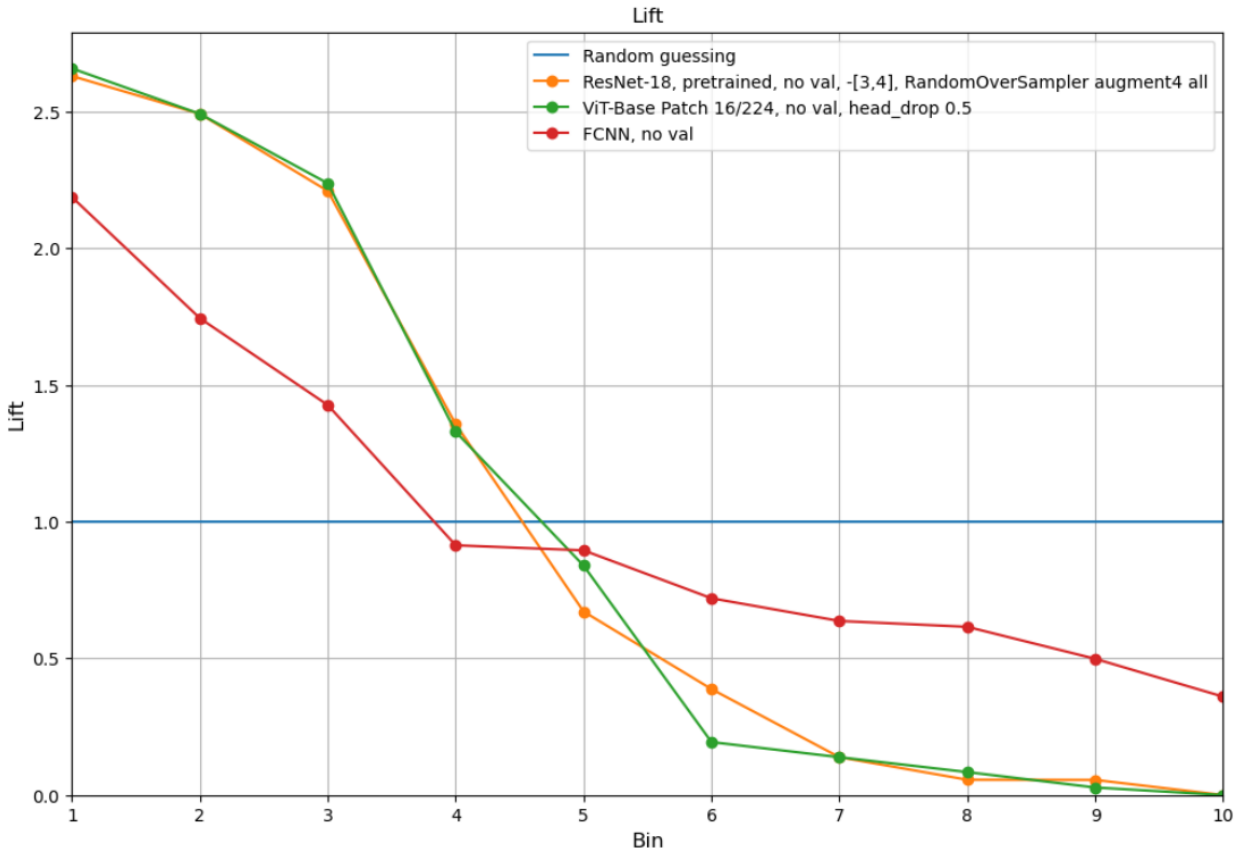
1.2.4 FCNN

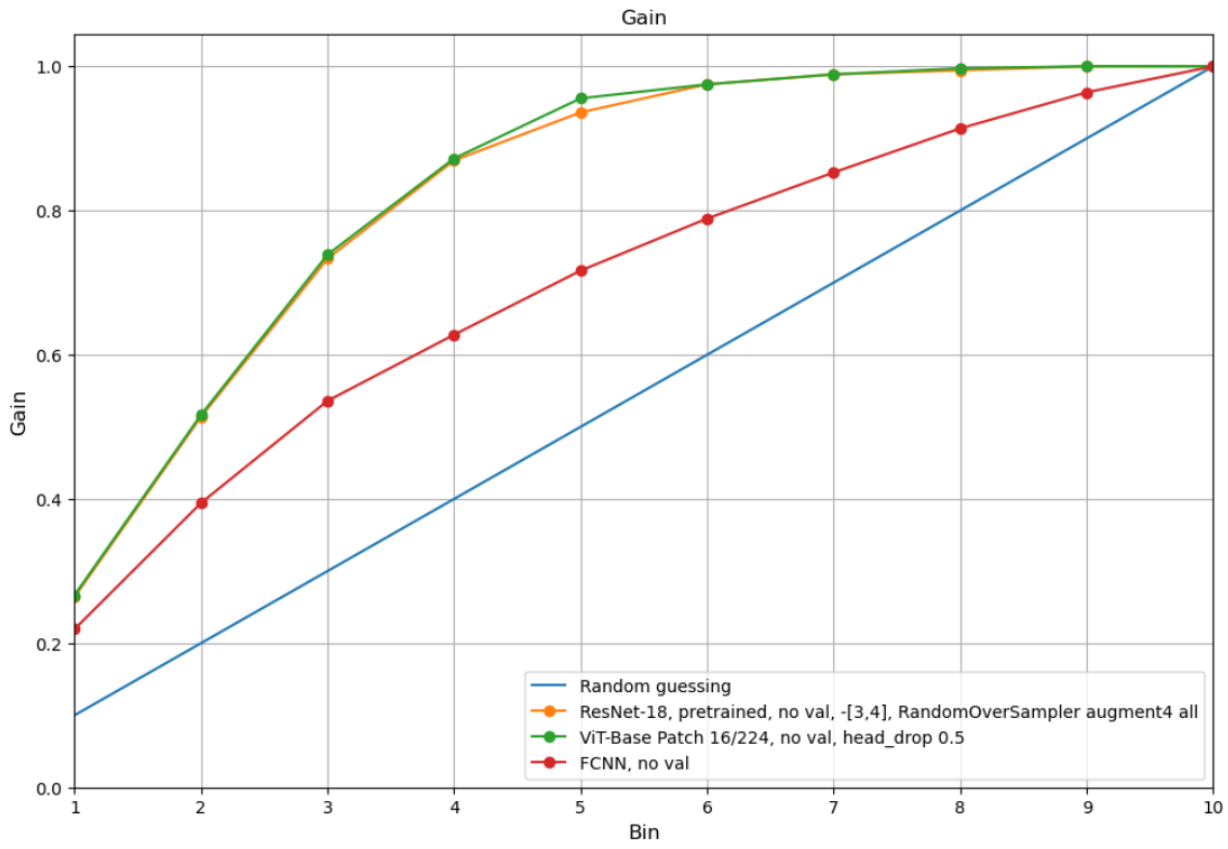
Training hyperparameters:

1. Single training/testing run
2. Randomly initialized weights; update all layers
3. No holdout validation
4. Remove 3/7 and 4/7 annotator images from training set
5. Random oversampling
6. Image augmentation on all samples
7. Normalized pixel values using mean and standard deviation of 0.5

Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
ResNet-18, pretrained, no val, -[3,4], RandomOverSampler augment4 all	13.717	0.861	0.825	0.922	0.843	0.886	0.874	0.942
ViT-Base Patch 16/224, no val, head_drop 0.5	73.056	0.832	0.842	0.901	0.837	0.879	0.871	0.948
FCNN, no val	19.160	0.641	0.581	0.810	0.609	0.726	0.695	0.743




Lift
Random guessing
ResNet-18, pretrained, no val, -[3,4], RandomOverSampler augment4 all
ViT-Base Patch 16/224, no val, head_drop 0.5
FCNN, no val
Lift
Bin
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7
8
9
10



1.3 Mistakes/issues

1. During ResNet experiments, I used validation data for triggering early stop, but I used testing data to select the best hyperparameters. I corrected this for the ViT experiments – I used the validation data to select the best hyperparameters and reported performance on the test data.
2. Did not set the random seed during ResNet experiments; set random seed to 42 for all ViT experiments for reproducibility.
3. Initially triggered early stop on AUROC (torchmetrics) but found that AUROC seems to be calculating incorrectly (see appendix). Moved to calculating AUC manually from ROC but finally settled on F1 score for computational efficiency.

1.4 Future work

1. Is there any more recent work on MHIST dataset?
2. Try freezing more layers to improve training speed
3. Try dropout and weight decay with ResNet-18
4. Try better transformer models (base transformer model has many drawbacks)
5. Systematic hyperparameter tuning using Optuna or Ray Tune
    a. Report training speed

6. Try hybrid CNN-ViT models (coat transformer)
7. Try GANs to generate synthetic images
8. Try to segment the region of interest – where is the cancer? https://github.com/bowang-lab/MedSAM/tree/MedSAM2. Unsupervised way of segmenting the images. Pass image through model; the output is a segment of the image.
9. Debug AUROC issue
10. Switch from functional programming to object-oriented programming
11. Switch from .ipynb to .py

2 ResNet-18

For computer vision, we spent the most time learning about CNNs, so that's why I chose to start with a CNN. The research paper also used ResNet-18.

Since this is binary classification, I changed the last layer from Linear(512, 1000) to Sigmoid(Linear(512, 1)). The cost function is binary cross entropy (torch.nn.BCELoss).

I used these training hyperparameters directly from the paper:

1. Initial learning rate of 0.001
2. Adam optimizer with a learning rate decay of 0.91 per epoch
3. Batch size = 32

To speed up training, I used early stopping, triggering on metrics calculated from the validation set and reloading the model state from the best epoch. I tried three different metrics from torchmetrics.

1. Since the research paper reported AUC, I started by triggering on AUROC, but I found a problem with AUROC (see appendix)
2. Then I used ROC and calculated AUC manually, but this was very slow
3. Finally, I used F1 score

Initially, I tried two configurations of ResNet-18:

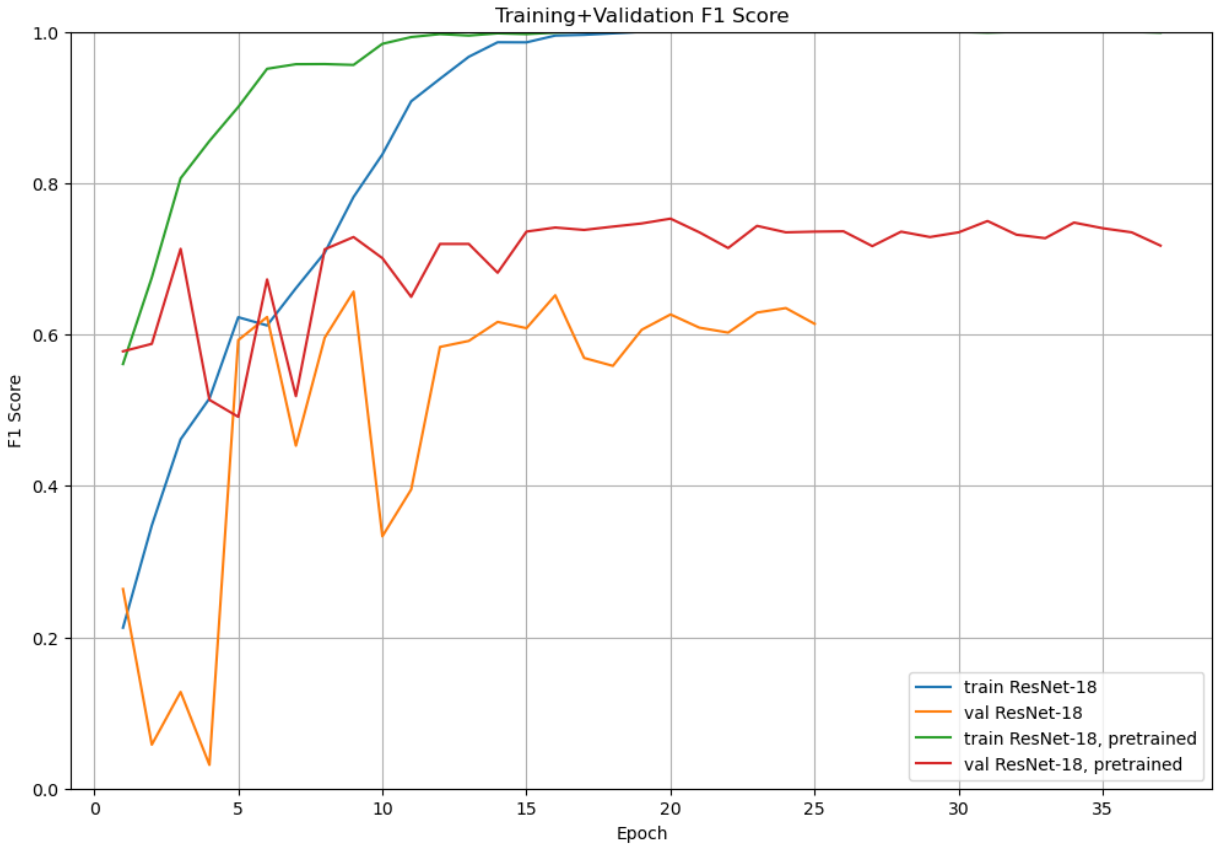
1. Random weight initialization, update all layers
2. Pretrained weights, update only final layer

Since updating only the final layer resulted in poor performance even for the training data, I decided to update all layers.

Pretrained weights improve performance, but the model is overfitting in both cases.

Training data:

Model	Best Epoch	Train Loss	Val Loss	Train AUC	Val AUC	Train F1	Val F1
ResNet-18	5	23.9	6.2	0.843	0.857	0.623	0.593
ResNet-18, pretrained	17	0.1	8.1	1.000	0.915	1.000	0.738



To improve generalization, I tried

1. Image augmentation
2. Removing difficult images – images where the annotators were split. The paper tried removing images with 3/7 and 4/7 annotators.
3. Class balancing

2.1 Random weight initialization

2.1.1 Image augmentation

I tried these augmentations:

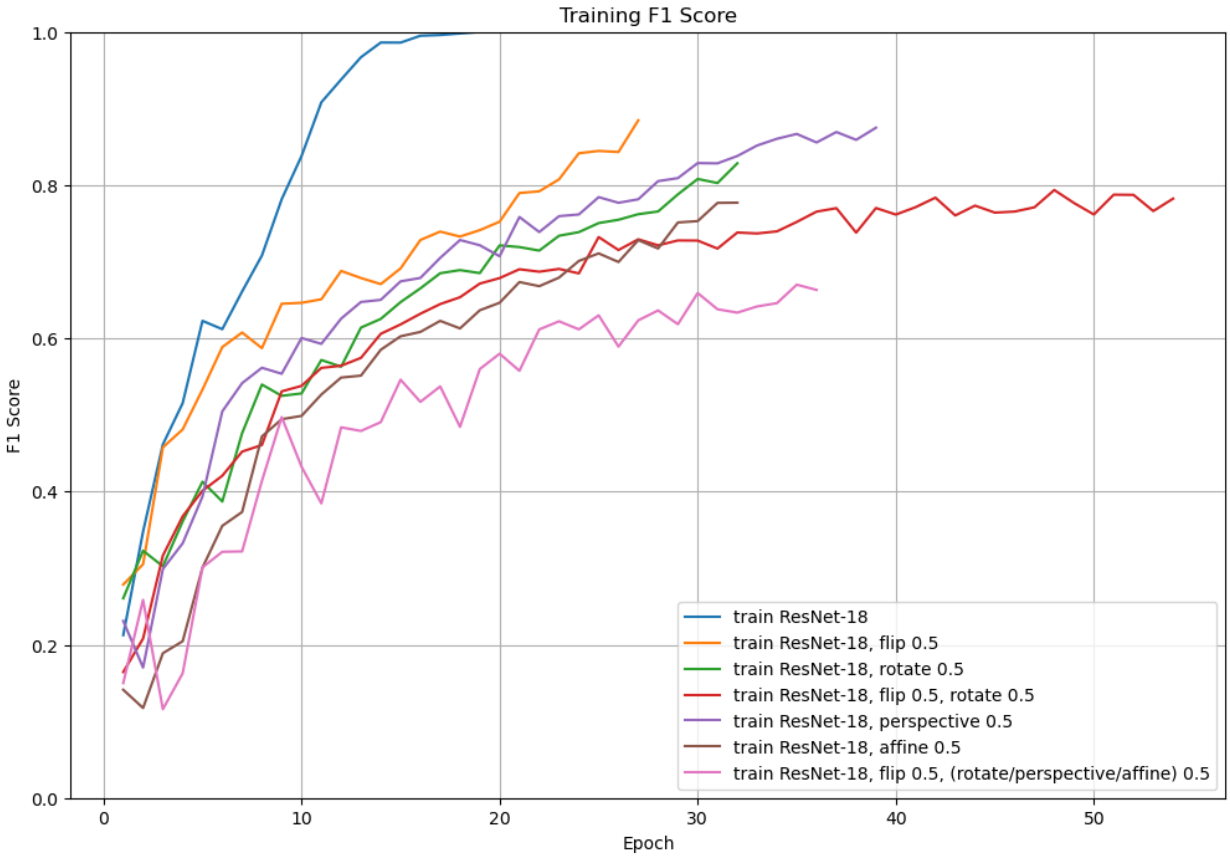
1. Horizontal flipping (p=0.5) + vertical flipping (p=0.5)
2. Rotation +/- 180 degrees with expansion and resizing to avoid cutting off part of the image, p=0.5
3. Flipping + rotation
4. Perspective, distortion=0.5, p=0.5
5. Affine (translate, shear, scale), p=0.5
6. Flipping + RandomChoice(rotation, perspective, affine)

This is a case where choosing the hyperparameter was affected by my choice to use testing data for tuning. Validation F1 score was best when using perspective, but testing AUC was best for flipping + rotation. I used flipping + rotation for further experiments.

In all cases, image augmentation reduces overfitting and improves generalization.

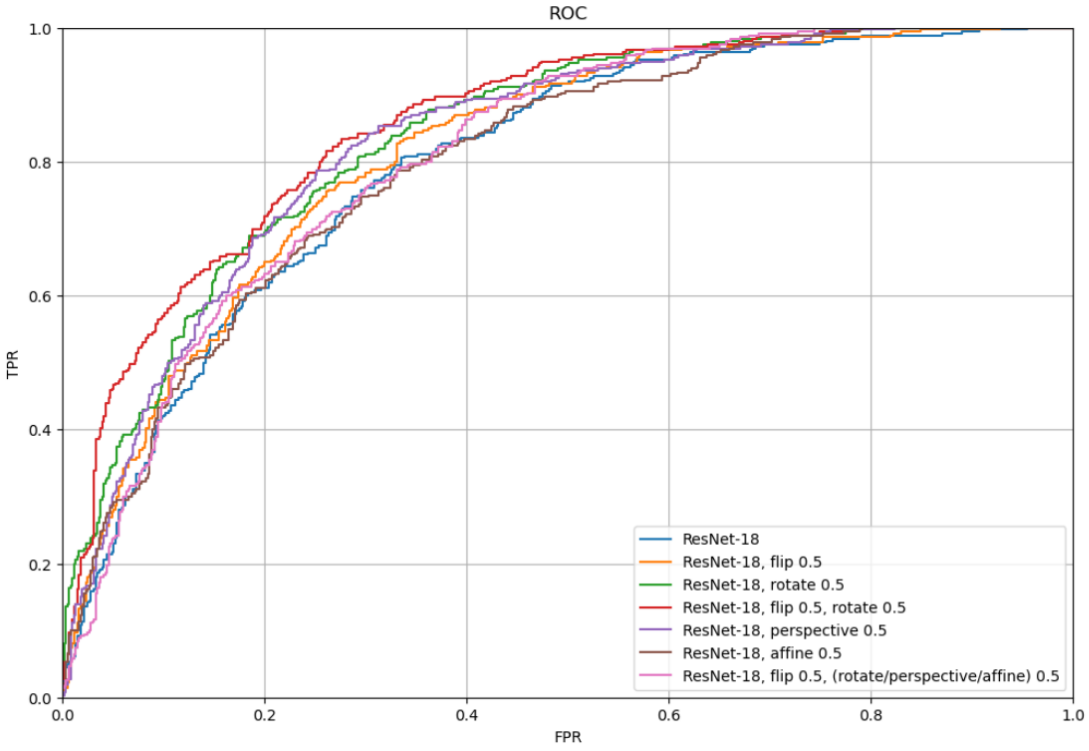
Training data:

Model	Best Epoch	Train Loss	Val Loss	Train AUC	Val AUC	Train F1	Val F1
ResNet-18	5	23.9	6.2	0.843	0.857	0.623	0.593
ResNet-18, flip 0.5	12	21.7	5.6	0.876	0.872	0.688	0.658
ResNet-18, rotate 0.5	17	20.8	5.5	0.887	0.880	0.685	0.642
ResNet-18, flip 0.5, rotate 0.5	34	17.5	5.2	0.922	0.902	0.740	0.694
ResNet-18, perspective 0.5	24	17.4	5.9	NaN	NaN	0.762	0.732
ResNet-18, affine 0.5	17	23.3	6.7	NaN	NaN	0.623	0.669
ResNet-18, flip 0.5, (rotate/perspective/affine) 0.5	21	24.0	5.9	NaN	NaN	0.558	0.682



Testing data:

Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
ResNet-18	31.572	0.661	0.564	0.831	0.609	0.733	0.698	0.798
ResNet-18, flip 0.5	20.762	0.702	0.492	0.878	0.578	0.736	0.685	0.816
ResNet-18, rotate 0.5	17.757	0.643	0.725	0.765	0.681	0.750	0.745	0.838
ResNet-18, flip 0.5, rotate 0.5	15.091	0.721	0.639	0.856	0.677	0.776	0.747	0.855
ResNet-18, perspective 0.5	20.999	0.678	0.644	0.822	0.661	0.756	0.733	0.831
ResNet-18, affine 0.5	20.764	0.703	0.486	0.880	0.575	0.735	0.683	0.800
ResNet-18, flip 0.5, (rotate/perspective/affine) 0.5	16.840	0.683	0.592	0.840	0.634	0.748	0.716	0.810



2.1.2 Removing difficult images

The research paper reported marginal improvement when removing difficult-to-classify images (images ranked 3/7 or 4/7 by annotators).

This idea is similar to Tomek Links, except Tomek Links only removes the majority class labels – in this case, 3/7.

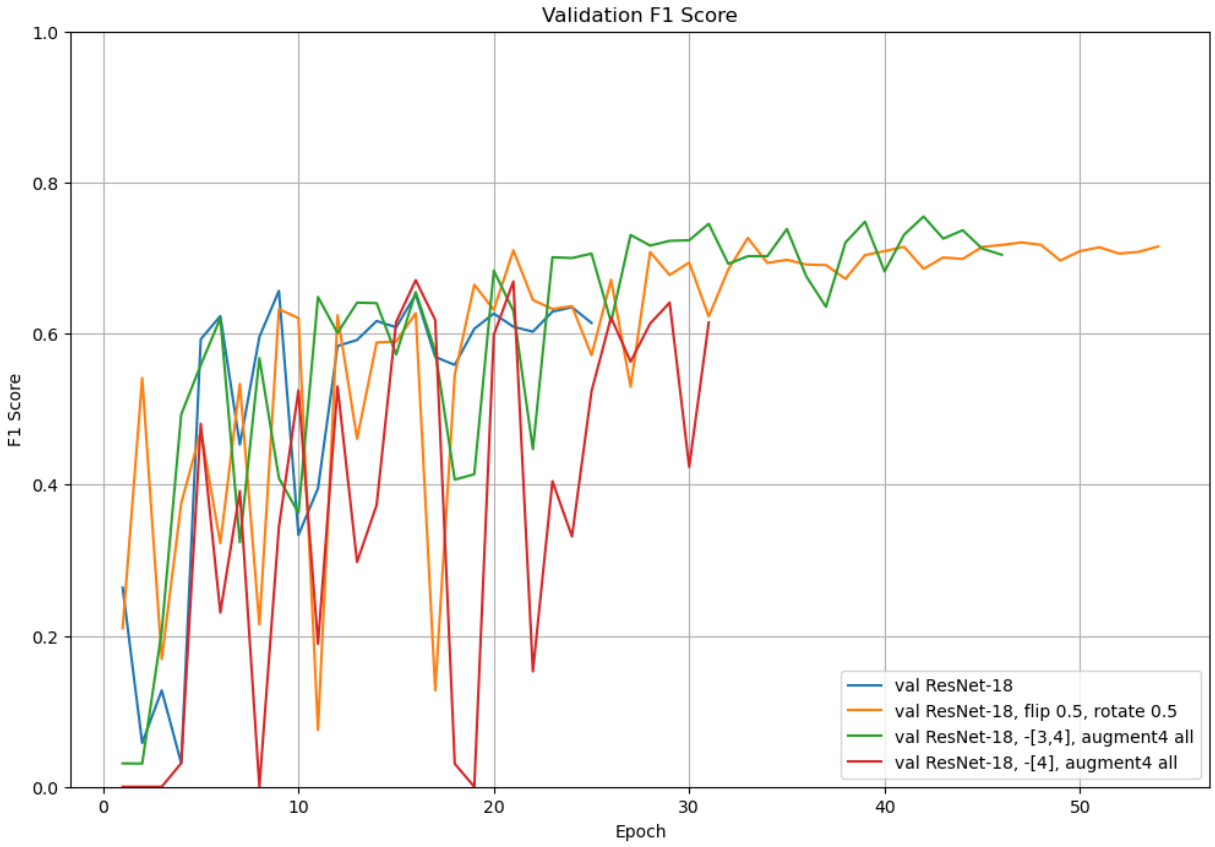
I tried removing 3/7, 4/7, and both 3/7 and 4/7. Removing 4/7 gives the best performance, but when combined with image augmentation, removing both 3/7 and 4/7 gives the best performance.

Training data, removing difficult images only:

Model	Best Epoch	Train Loss	Val Loss	Train AUC	Val AUC	Train F1	Val F1
ResNet-18	5	23.9	6.2	0.843	0.857	0.623	0.593
ResNet-18, -[3]	14	4.4	14.5	NaN	NaN	0.947	0.650
ResNet-18, -[3,4]	16	1.7	10.8	NaN	NaN	0.976	0.669
ResNet-18, -[4]	10	12.9	6.3	NaN	NaN	0.778	0.679

Training data, removing difficult images and image augmentation:

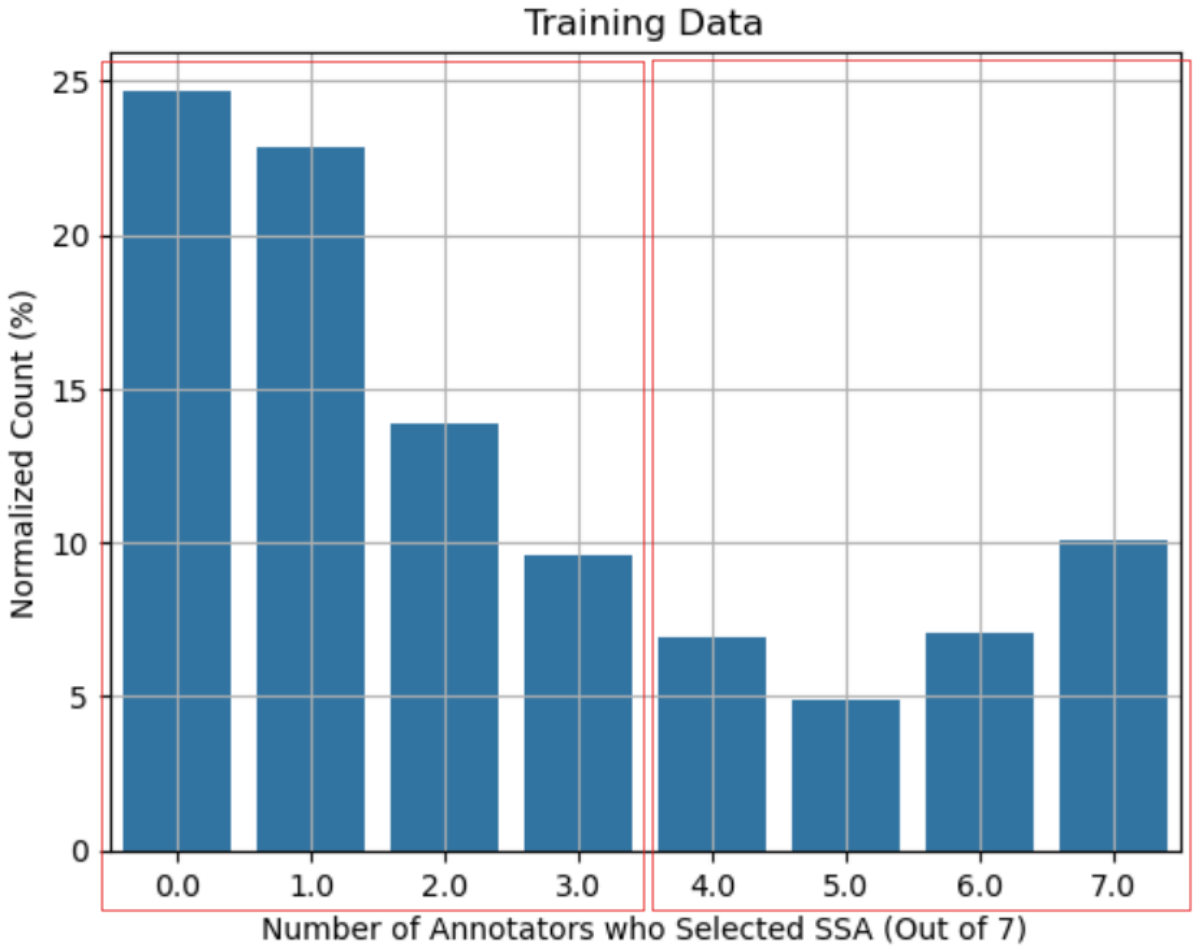
Model	Best Epoch	Train Loss	Val Loss	Train AUC	Val AUC	Train F1	Val F1
ResNet-18	5	23.9	6.2	0.843	0.857	0.623	0.593
ResNet-18, flip 0.5, rotate 0.5	34	17.5	5.2	0.922	0.902	0.740	0.694
ResNet-18, -[3,4], augment4 all	31	12.4	5.1	0.942	0.907	0.756	0.745
ResNet-18, -[4], augment4 all	16	20.6	6.7	NaN	NaN	0.548	0.671



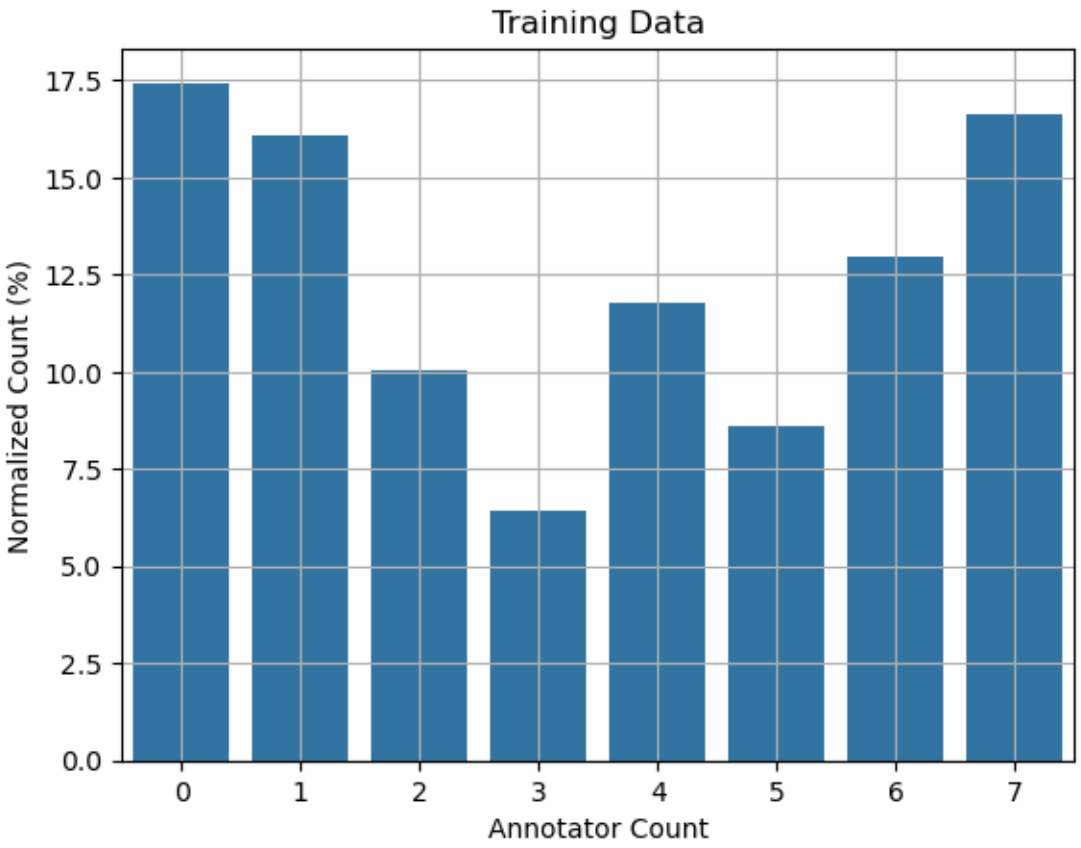
2.1.3 Class balancing

I tried random oversampling, SMOTE, and ADASYN. In each epoch, I sampled every sample at least once. Each epoch trains on 2*(number of majority class samples); to make up the difference of the minority class, I uniformly re-sampled minority class samples. For ADASYN, I uniformly re-sampled the most difficult-to-classify minority samples.

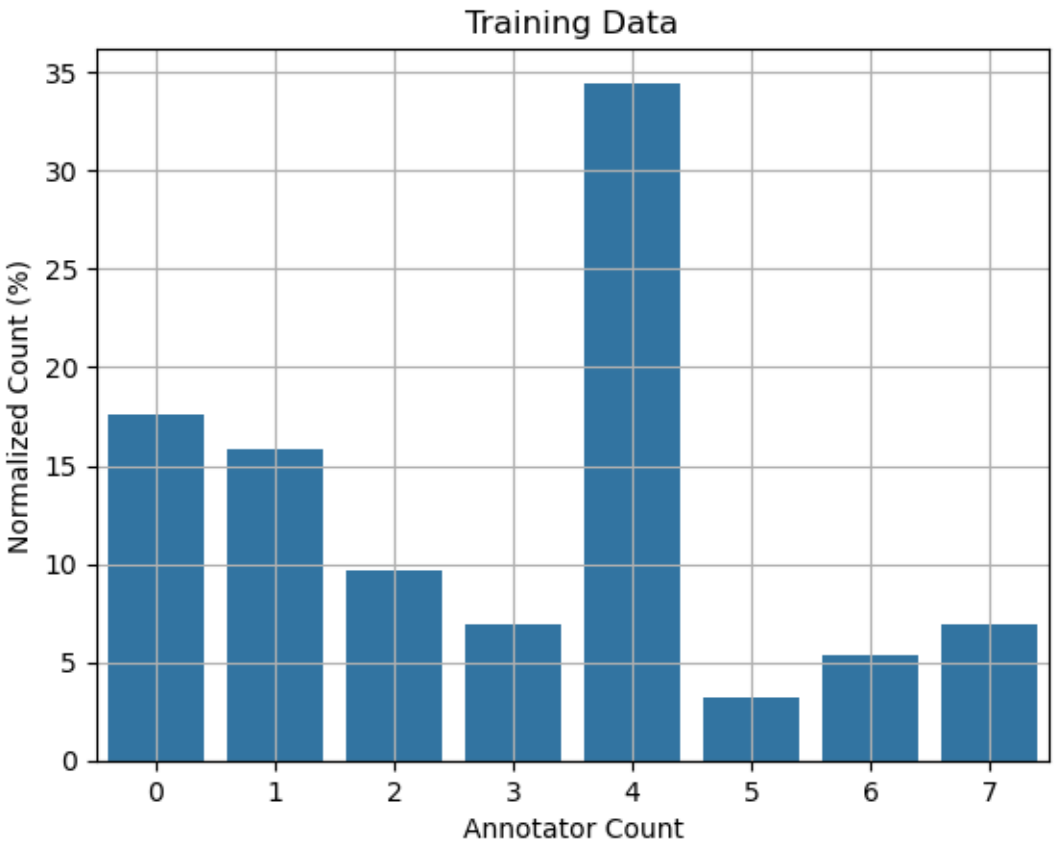
Baseline annotator distribution:



Annotator distribution for random oversampling and SMOTE:



Annotator distribution for ADASYN:

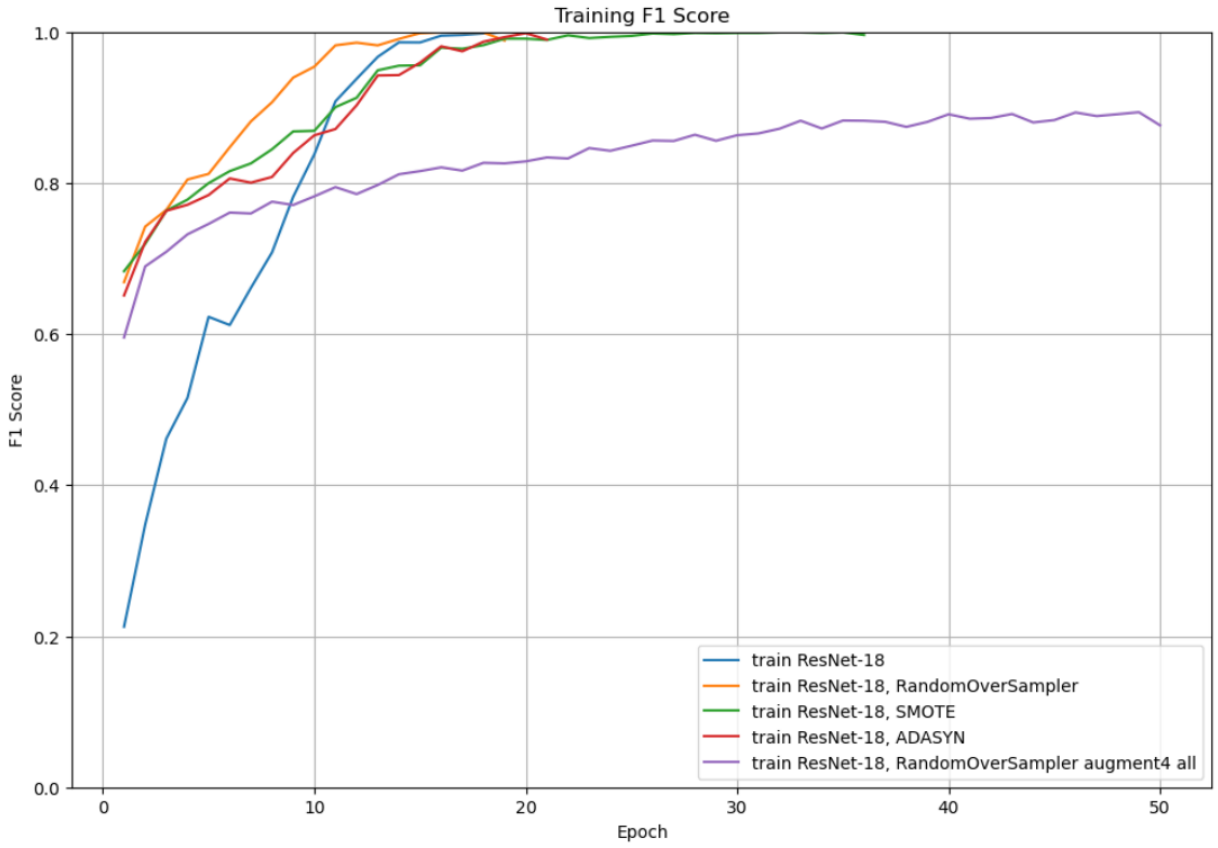


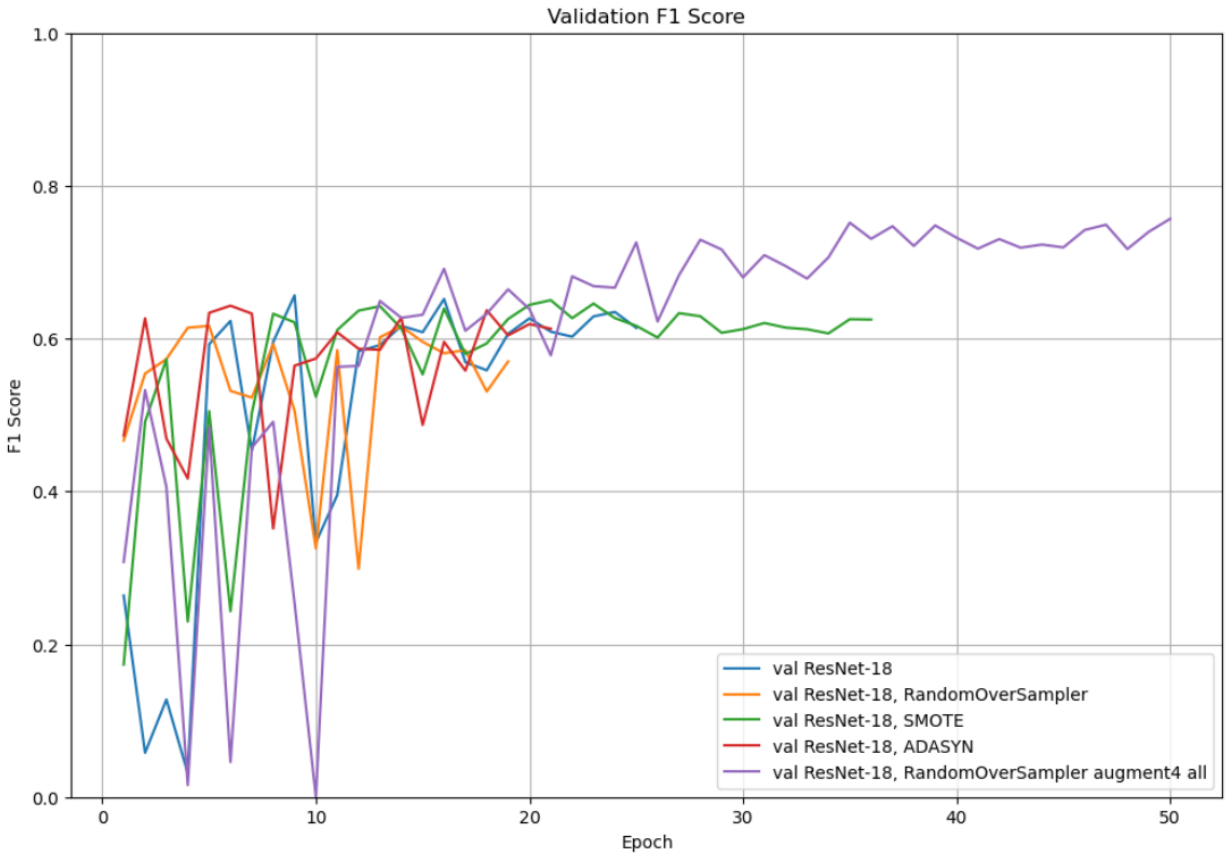
For SMOTE and ADASYN, instead of generating synthetic images, I applied image augmentation to generate “new” samples.

Simply balancing the classes, via random oversampling, doesn’t reduce overfitting or improve generalization. SMOTE and ADASYN help, but the best performance is given by balancing the classes and augmenting all samples.

Training data:

Model	Best Epoch	Train Loss	Val Loss	Train AUC	Val AUC	Train F1	Val F1
ResNet-18	5	23.9	6.2	0.843	0.857	0.623	0.593
ResNet-18, RandomOverSampler	4	35.5	7.9	NaN	NaN	0.805	0.614
ResNet-18, SMOTE	21	2.6	15.2	NaN	NaN	0.989	0.650
ResNet-18, ADASYN	6	31.5	7.1	NaN	NaN	0.806	0.643
ResNet-18, RandomOverSampler augment4 all	35	22.8	5.6	NaN	NaN	0.883	0.752


Training F1 Score
F1 Score
Epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
train ResNet-18
train ResNet-18, RandomOverSampler
train ResNet-18, SMOTE
train ResNet-18, ADASYN
train ResNet-18, RandomOverSampler augment4 all



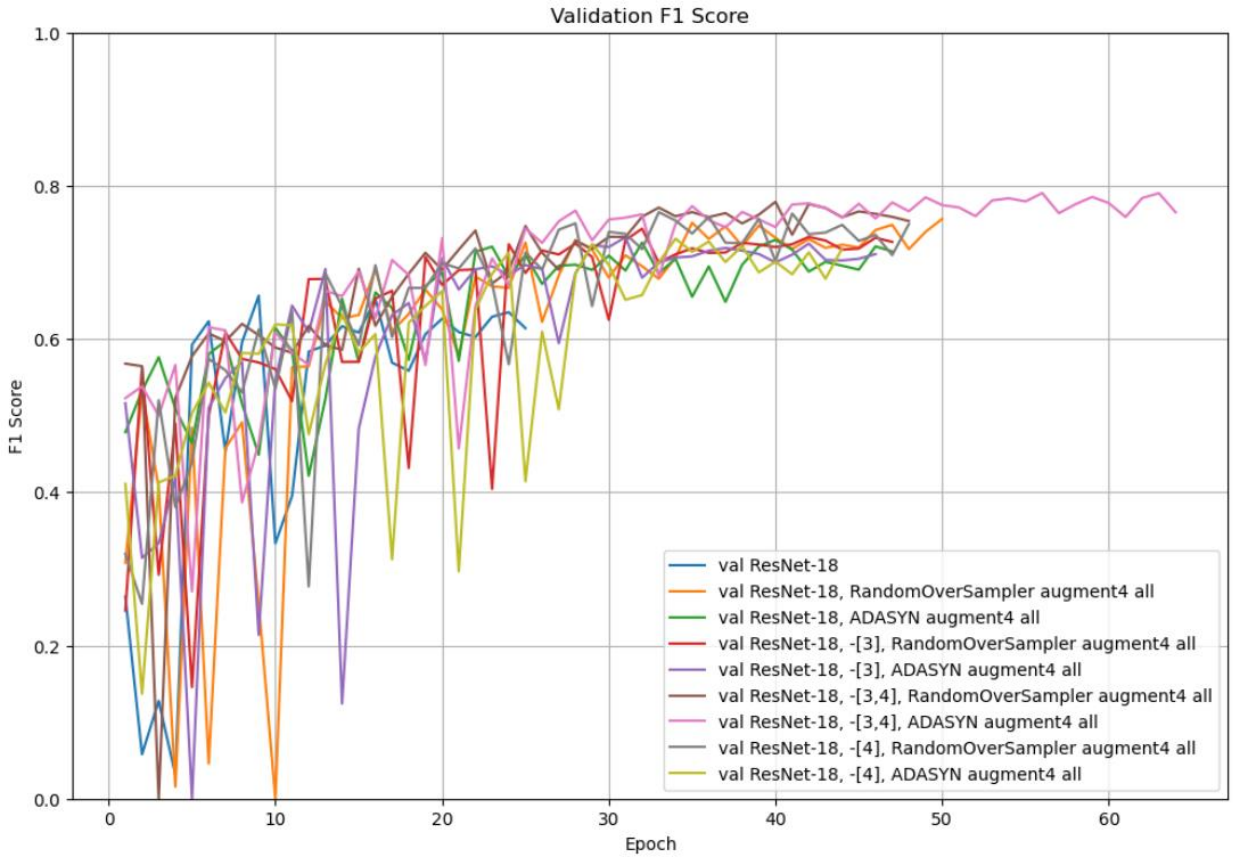
2.1.3.1 Random oversampling vs. ADASYN

Since augmenting all samples gives the best performance, I next tried combinations of oversampling techniques (random oversampling vs. ADASYN) and removing difficult samples.

This is a case where choosing the hyperparameter was affected by my choice to use testing data for tuning. Validation F1 score was best with ADASYN and removing 3/7 and 4/7, but testing AUC was best for random oversampling and removing 3/7 and 4/7, which I used for further experiments.

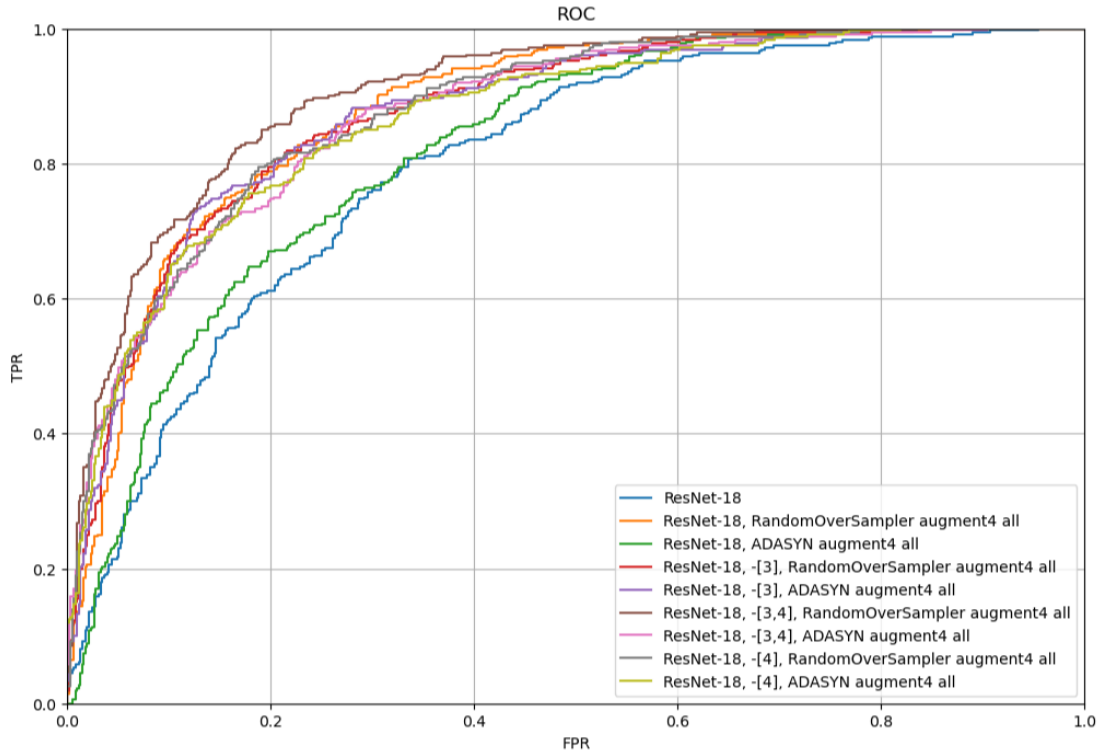
Training data:

Model	Best Epoch	Train Loss	Val Loss	Train AUC	Val AUC	Train F1	Val F1
ResNet-18	5	23.9	6.2	0.843	0.857	0.623	0.593
ResNet-18, RandomOverSampler augment4 all	35	22.8	5.6	NaN	NaN	0.883	0.752
ResNet-18, ADASYN augment4 all	32	24.4	5.8	NaN	NaN	0.869	0.726
ResNet-18, -[3], RandomOverSampler augment4 all	32	17.8	5.7	NaN	NaN	0.893	0.744
ResNet-18, -[3], ADASYN augment4 all	31	17.0	6.8	NaN	NaN	0.897	0.733
ResNet-18, -[3,4], RandomOverSampler augment4 all	33	12.3	5.8	NaN	NaN	0.929	0.772
ResNet-18, -[3,4], ADASYN augment4 all	49	8.0	6.5	NaN	NaN	0.958	0.785
ResNet-18, -[4], RandomOverSampler augment4 all	33	18.2	5.5	NaN	NaN	0.910	0.766
ResNet-18, -[4], ADASYN augment4 all	29	16.4	7.1	NaN	NaN	0.918	0.724



Testing data:

Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
ResNet-18	31.572	0.661	0.564	0.831	0.609	0.733	0.698	0.798
ResNet-18, RandomOverSampler augment4 all	13.991	0.732	0.753	0.840	0.742	0.808	0.796	0.880
ResNet-18, ADASYN augment4 all	20.871	0.678	0.639	0.823	0.658	0.755	0.731	0.821
ResNet-18, -[3], RandomOverSampler augment4 all	16.382	0.681	0.819	0.776	0.744	0.792	0.798	0.876
ResNet-18, -[3], ADASYN augment4 all	16.715	0.694	0.800	0.794	0.743	0.796	0.797	0.875
ResNet-18, -[3,4], RandomOverSampler augment4 all	14.847	0.787	0.717	0.887	0.750	0.824	0.802	0.906
ResNet-18, -[3,4], ADASYN augment4 all	18.588	0.754	0.672	0.872	0.711	0.798	0.772	0.873
ResNet-18, -[4], RandomOverSampler augment4 all	16.060	0.732	0.719	0.846	0.725	0.799	0.783	0.878
ResNet-18, -[4], ADASYN augment4 all	17.421	0.748	0.683	0.865	0.714	0.798	0.774	0.869



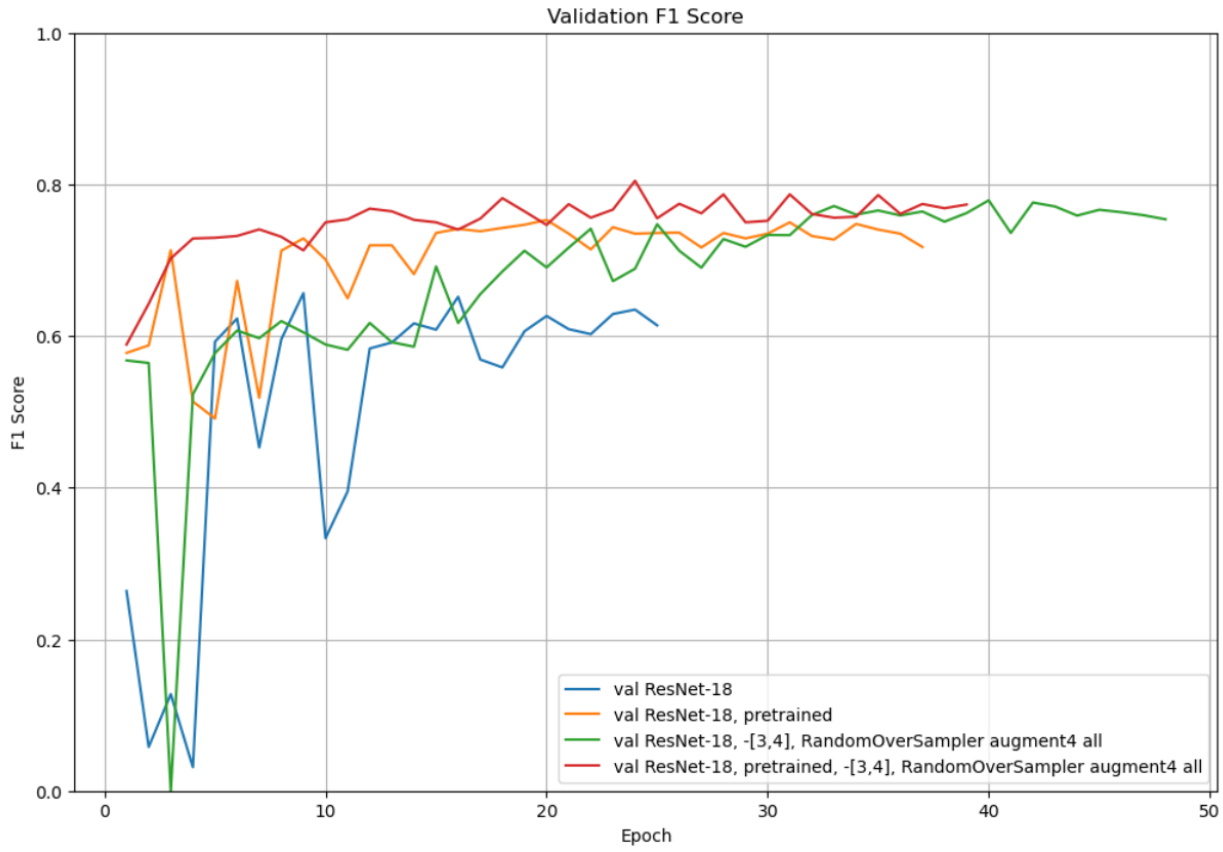
2.2 Pretrained weights

The trends for pretrained weights are the same, but there's less improvement with generalization techniques, because pretraining already helps with generalization. The research paper observes that as dataset size increases, the gain from pretraining decreases.

For our dataset, pretraining significantly boosts performance over randomly initialized weights.

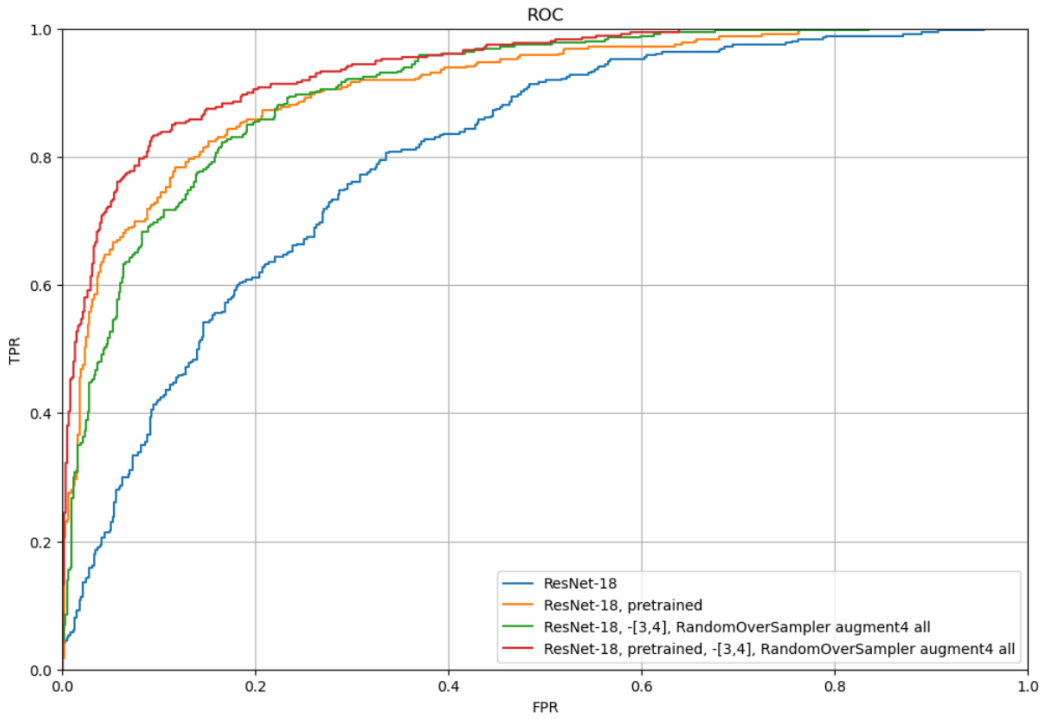
Training results:

Model	Best Epoch	Train Loss	Val Loss	Train AUC	Val AUC	Train F1	Val F1
ResNet-18	5	23.9	6.2	0.843	0.857	0.623	0.593
ResNet-18, pretrained	17	0.1	8.1	1.000	0.915	1.000	0.738
ResNet-18, -[3,4], RandomOverSampler augment4 all	33	12.3	5.8	NaN	NaN	0.929	0.772
ResNet-18, pretrained, -[3,4], RandomOverSampler augment4 all	24	4.4	7.2	NaN	NaN	0.977	0.805



Testing results:

Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
ResNet-18	31.572	0.661	0.564	0.831	0.609	0.733	0.698	0.798
ResNet-18, pretrained	23.592	0.873	0.667	0.943	0.756	0.841	0.805	0.909
ResNet-18, -[3,4], RandomOverSampler augment4 all	14.847	0.787	0.717	0.887	0.750	0.824	0.802	0.906
ResNet-18, pretrained, -[3,4], RandomOverSampler augment4 all	14.074	0.815	0.842	0.888	0.828	0.871	0.865	0.937

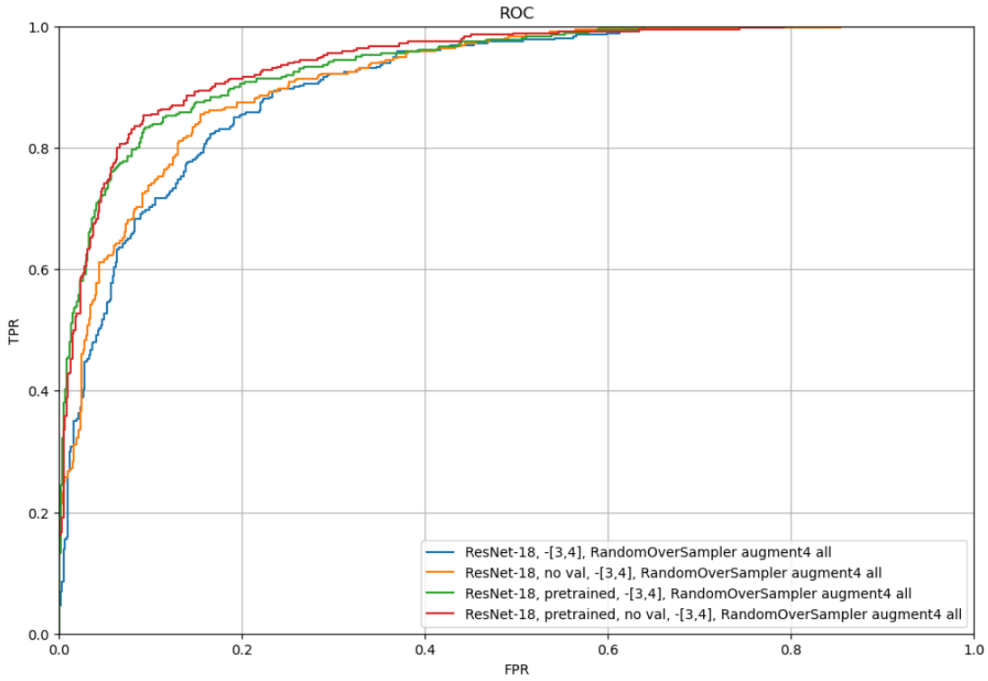


2.3 No validation

The final step is to train with the entire dataset. Training with the entire dataset improves performance.

Testing data:

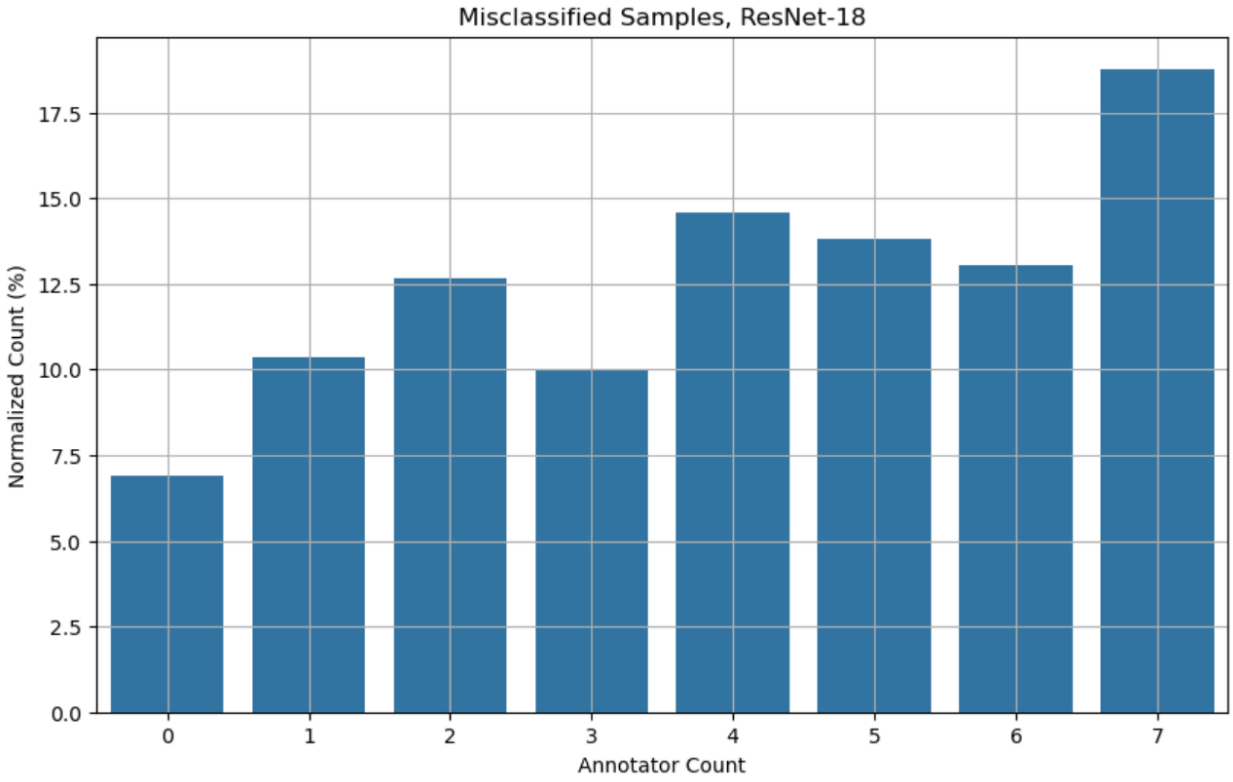
Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
ResNet-18, -[3,4], RandomOverSampler augment4 all	14.847	0.787	0.717	0.887	0.750	0.824	0.802	0.906
ResNet-18, no val, -[3,4], RandomOverSampler augment4 all	12.995	0.795	0.764	0.885	0.779	0.840	0.824	0.916
ResNet-18, pretrained, -[3,4], RandomOverSampler augment4 all	14.074	0.815	0.842	0.888	0.828	0.871	0.865	0.937
ResNet-18, pretrained, no val, -[3,4], RandomOverSampler augment4 all	13.717	0.861	0.825	0.922	0.843	0.886	0.874	0.942

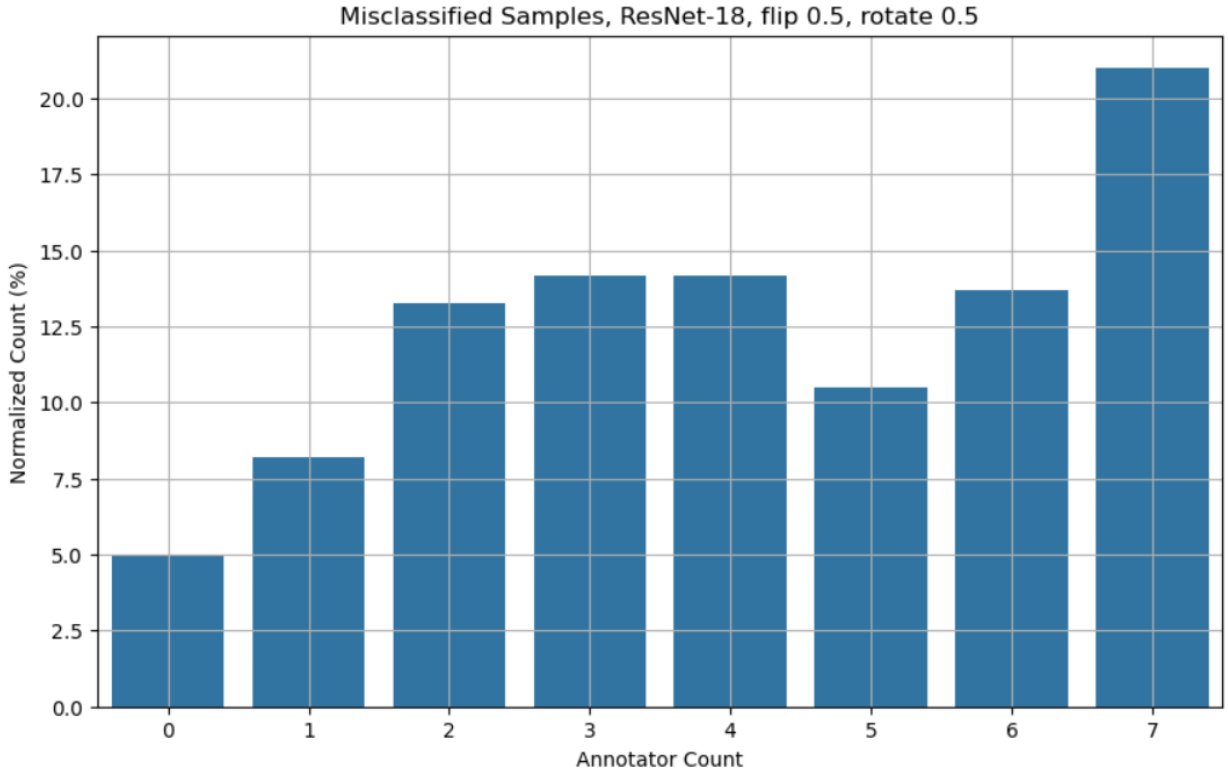


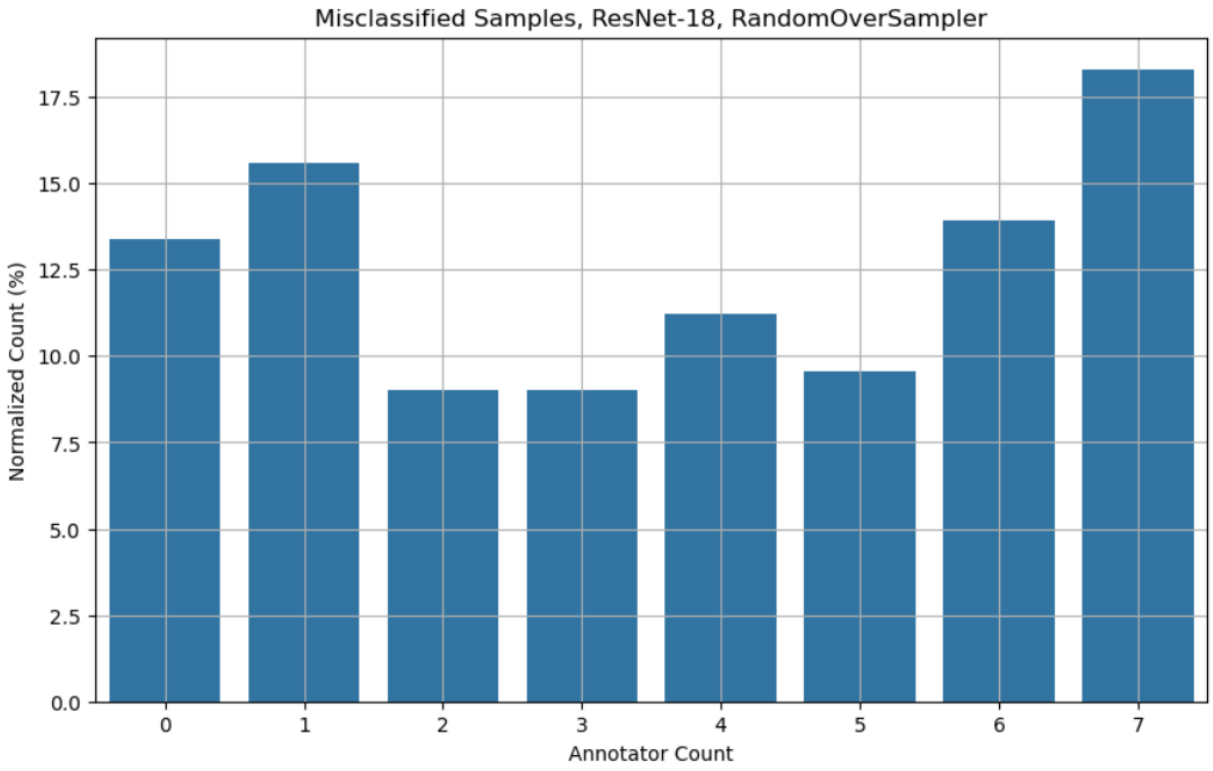
2.4 Misclassified samples

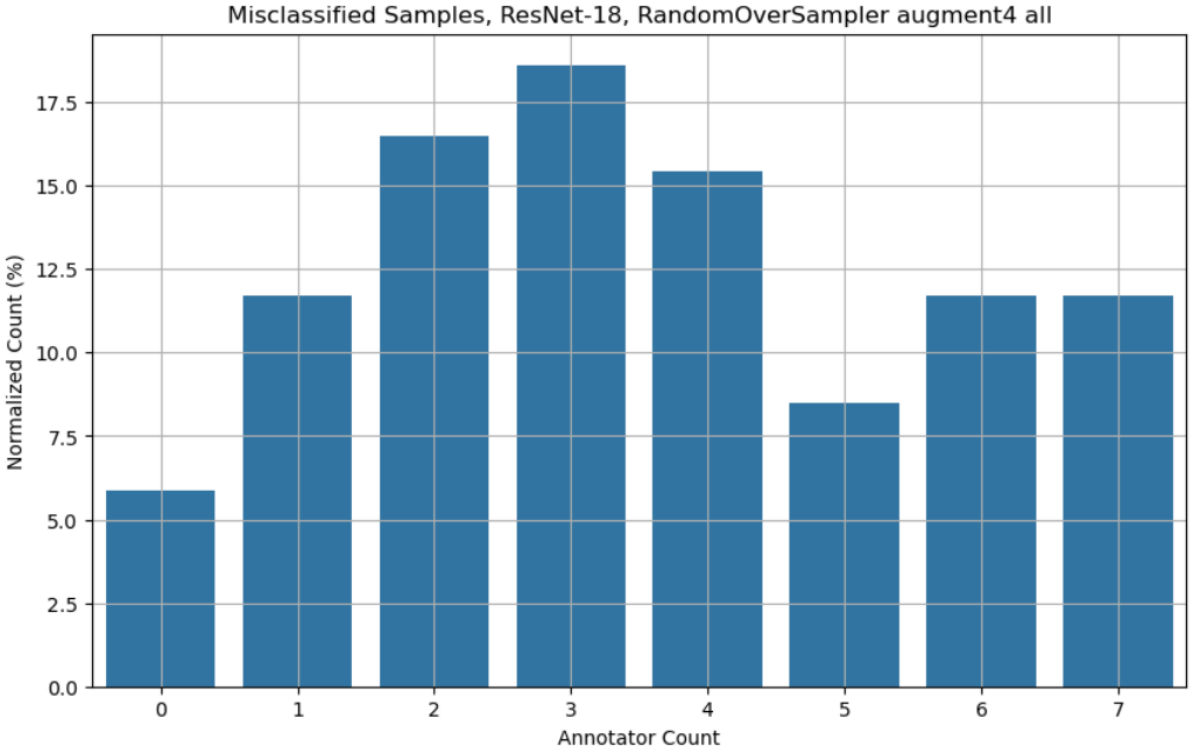
For both random weight initialization and pretrained weights, the misclassified samples skew towards higher number of annotators, with 7/7 images being the most misclassified class. Adding class balancing and data augmentation helps reduce this skew and changes the distribution to look more Gaussian, centered at 3/7 and 4/7.

2.4.1 Random weight initialization

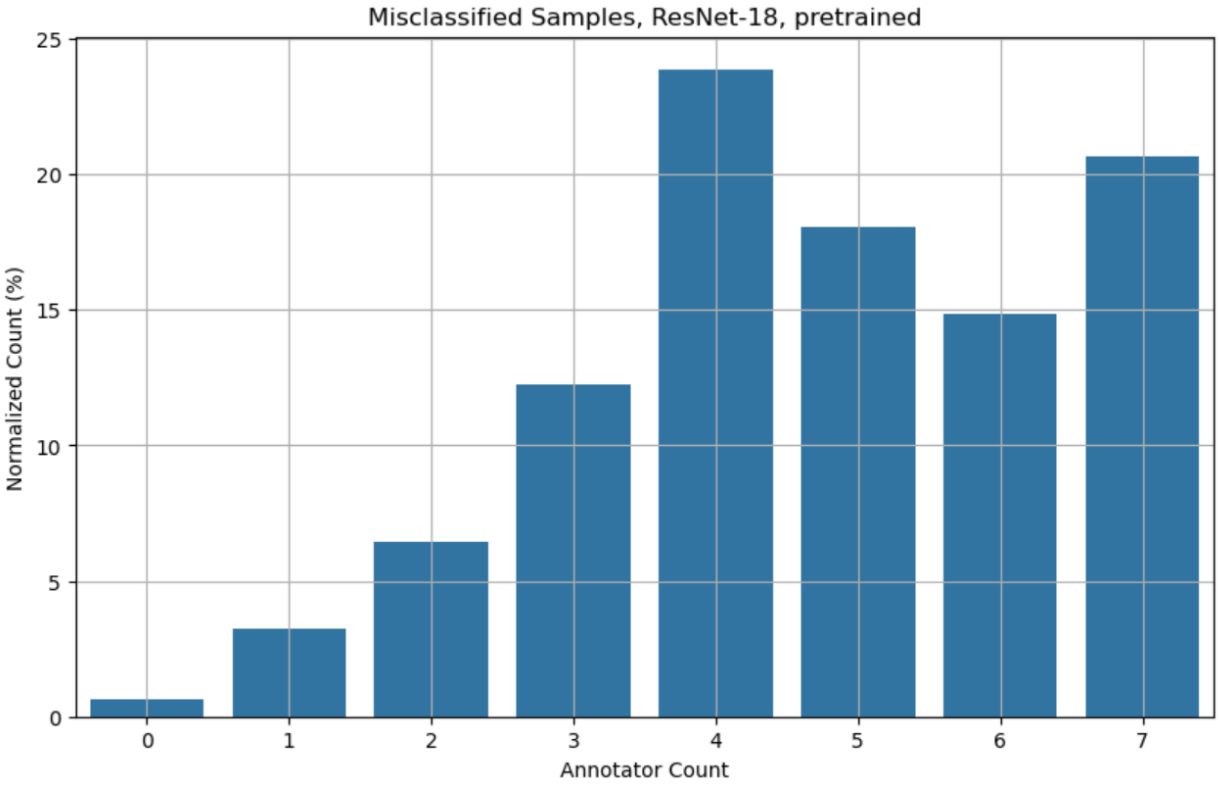


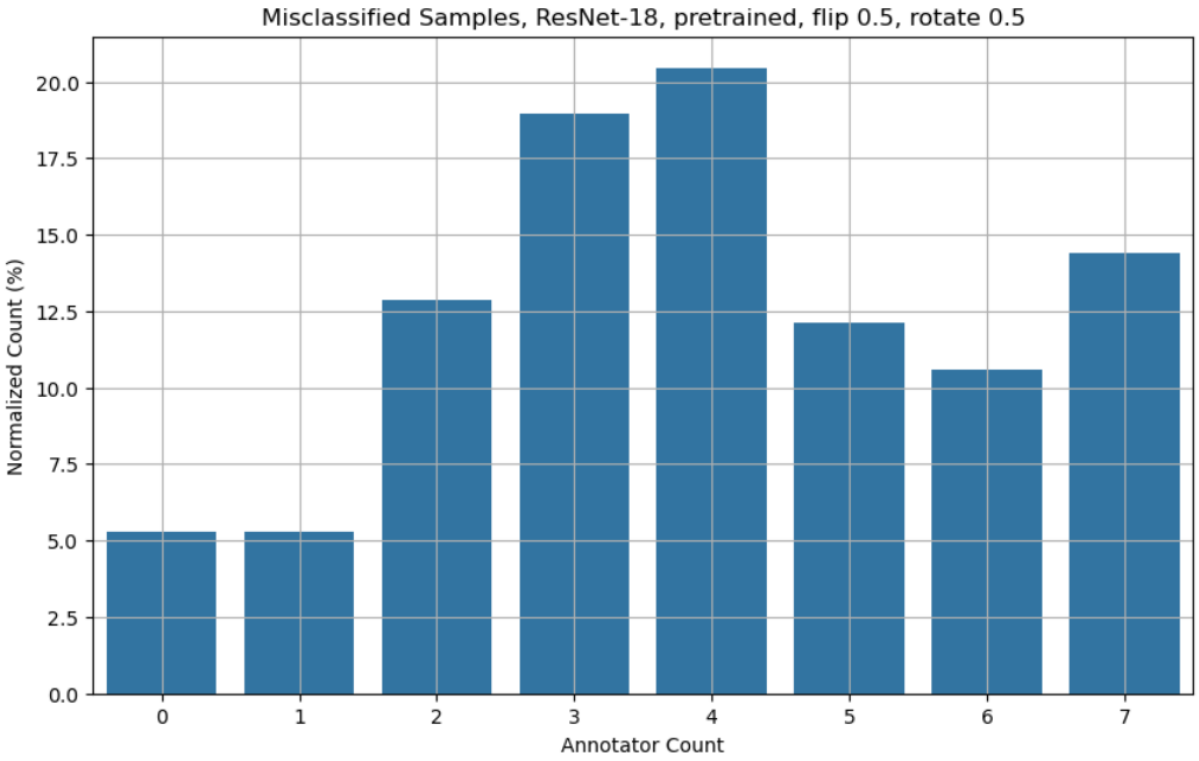


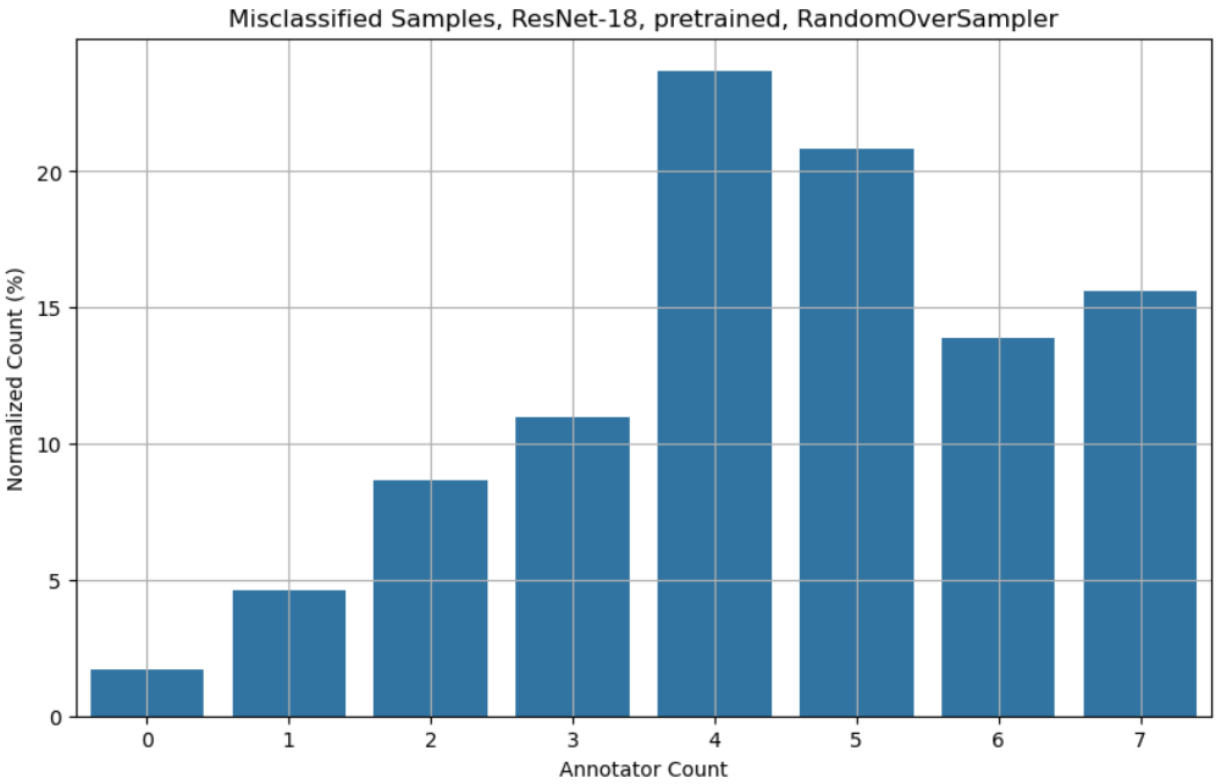

Misclassified Samples, ResNet-18, RandomOverSampler
Normalized Count (%)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
1
2
3
4
5
6
7
Annotator Count

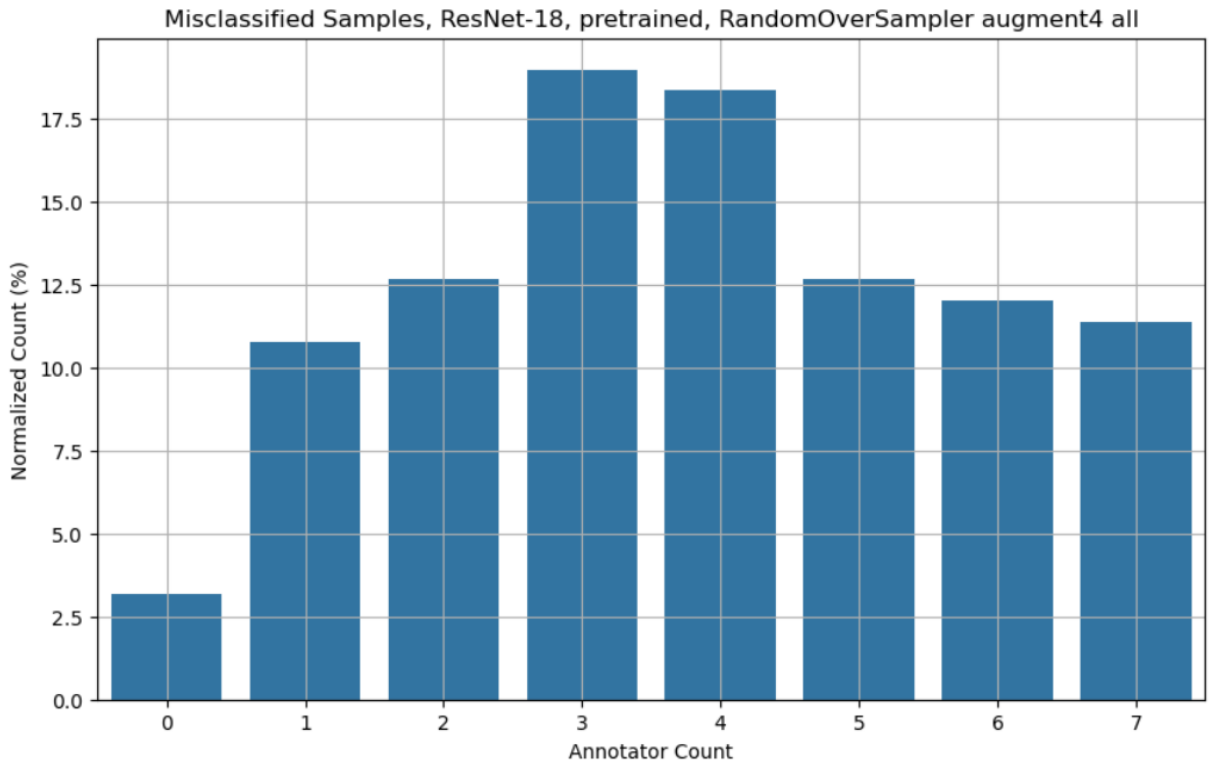

Misclassified Samples, ResNet-18, RandomOverSampler augment4 all
Normalized Count (%)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
1
2
3
4
5
6
7
Annotator Count

2.4.2 Pretrained weights






Misclassified Samples, ResNet-18, pretrained, RandomOverSampler
Normalized Count (%)
0
5
10
15
20
0
1
2
3
4
5
6
7
Annotator Count


Misclassified Samples, ResNet-18, pretrained, RandomOverSampler augment4 all
Normalized Count (%)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
1
2
3
4
5
6
7
Annotator Count

3 ViT-Base Patch 16/224

I took the best training hyperparameters from my ResNet-18 experiments:

1. Pretrained weights; update all layers
2. Remove 3/7 and 4/7 annotator images from training set
3. Random oversampling
4. Image augmentation on all samples

And then I tried tuning effective batch size, learning rate, dropout, and weight decay.

3.1 Effective batch size

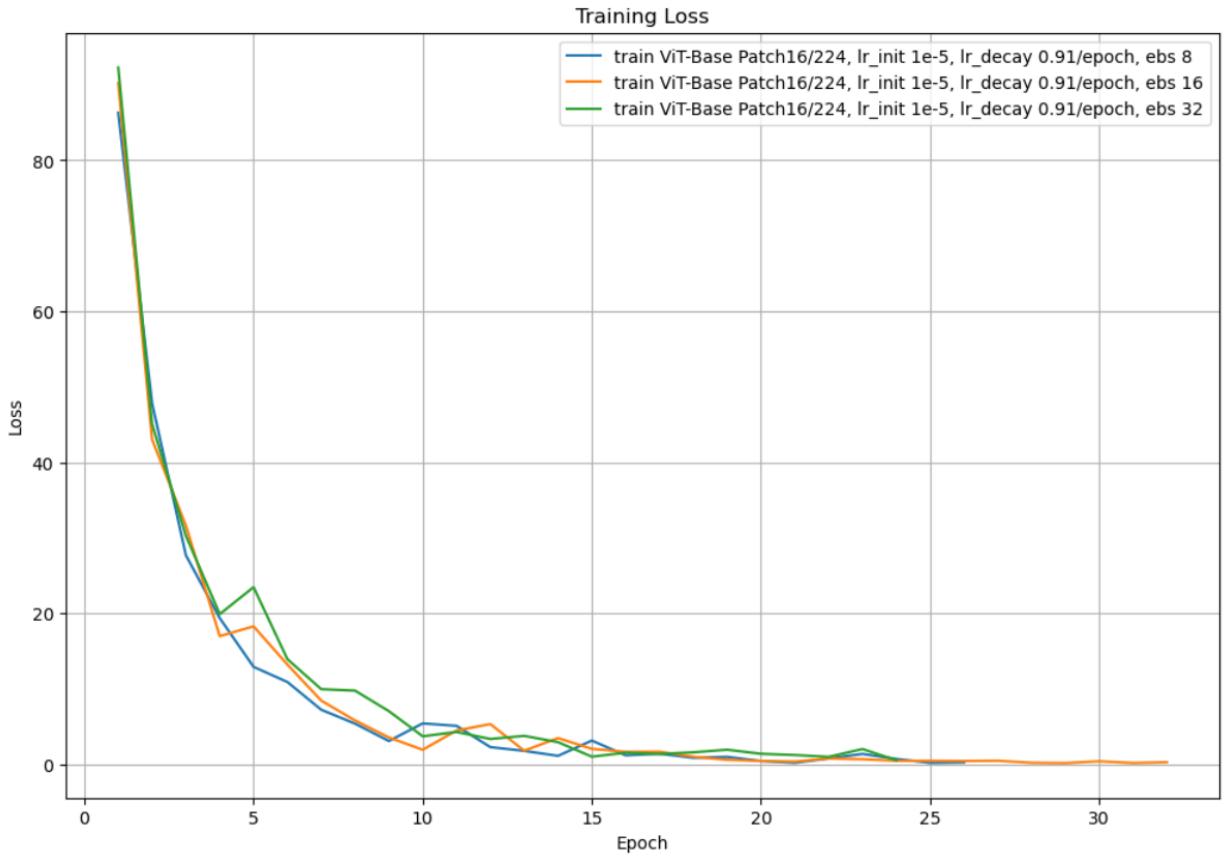
Since this vision transformer requires more memory than ResNet, I reduced the batch size from 32 to 8 and monitored my GPU memory usage:

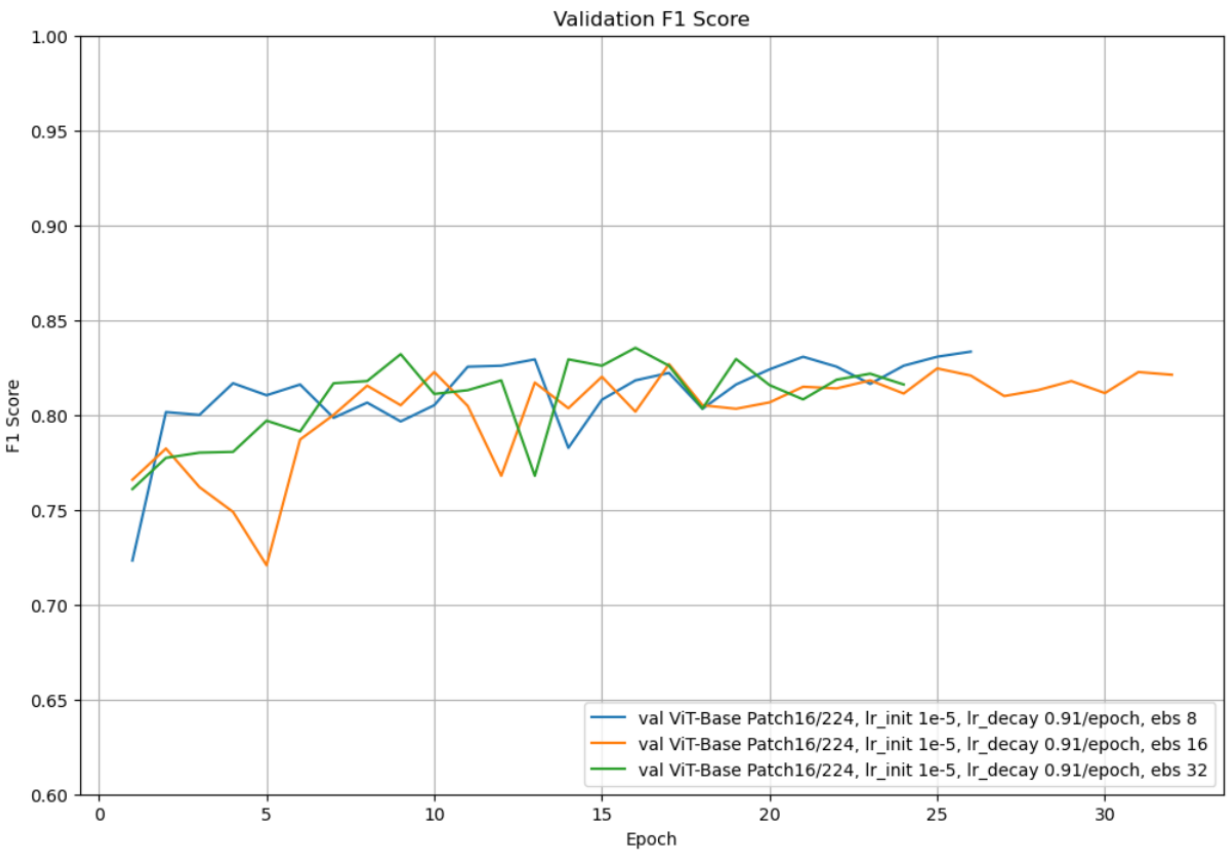
1. ResNet-18 with batch size 32: 2.3GB / 6.0GB
2. ViT-Base Patch 16/224 with batch size 8: 3.7GB / 6.0GB

I added gradient accumulation so I could try different effective batch sizes (effective batch size = batch size * number of batches to accumulate per update). I found that effective batch size didn't make much of a difference in either training convergence or validation F1 score, so I used an effective batch size of 8 for all further experiments.

Training results:

Model	Best Epoch	Train Loss	Val Loss	Train AUC	Val AUC	Train F1	Val F1
ViT-Base Patch16/224, lr_init 1e-5, lr_decay 0.91/epoch, ebs 8	11	5.1	30.1	NaN	NaN	0.993	0.825
ViT-Base Patch16/224, lr_init 1e-5, lr_decay 0.91/epoch, ebs 16	17	1.7	34.2	NaN	NaN	0.998	0.827
ViT-Base Patch16/224, lr_init 1e-5, lr_decay 0.91/epoch, ebs 32	9	7.1	26.2	NaN	NaN	0.992	0.832





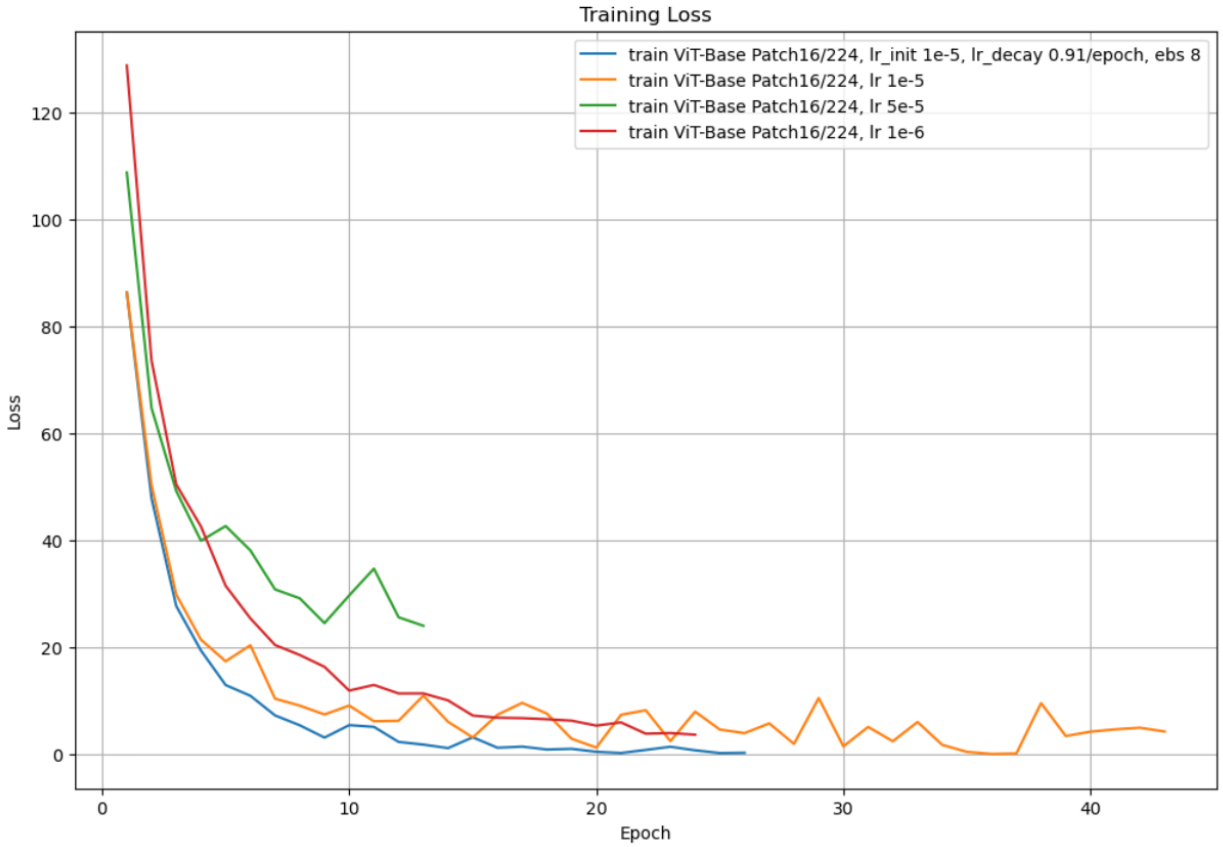
3.2 Learning rate

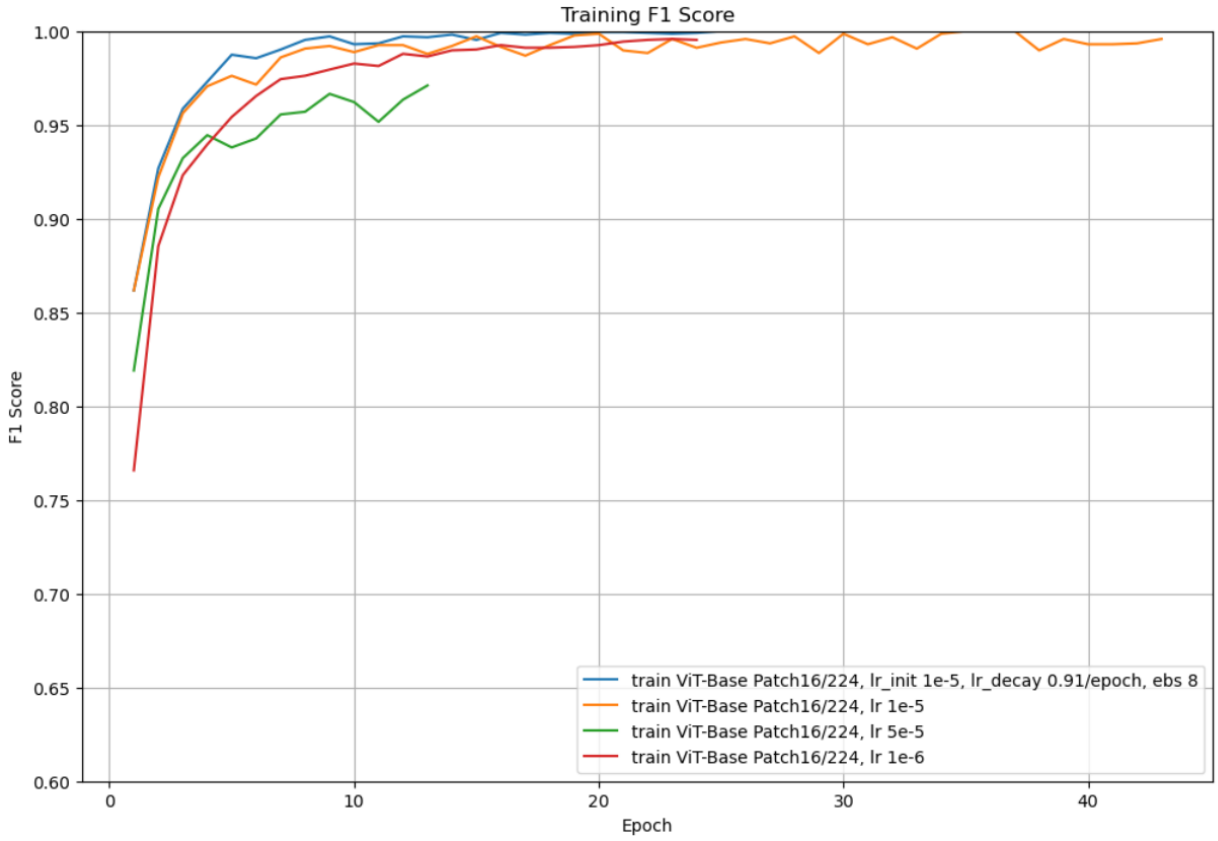
Using the same learning rate and scheduler as ResNet-18 caused training loss to blow up, so as a guess, I reduced the initial learning rate from 1e-3 to 1e-5, and training converged. I experimented with a few static learning rates but didn't see any obvious improvements to make. For all further experiments, I used an initial learning rate of 1e-5, decaying at a rate of 0.91/epoch.

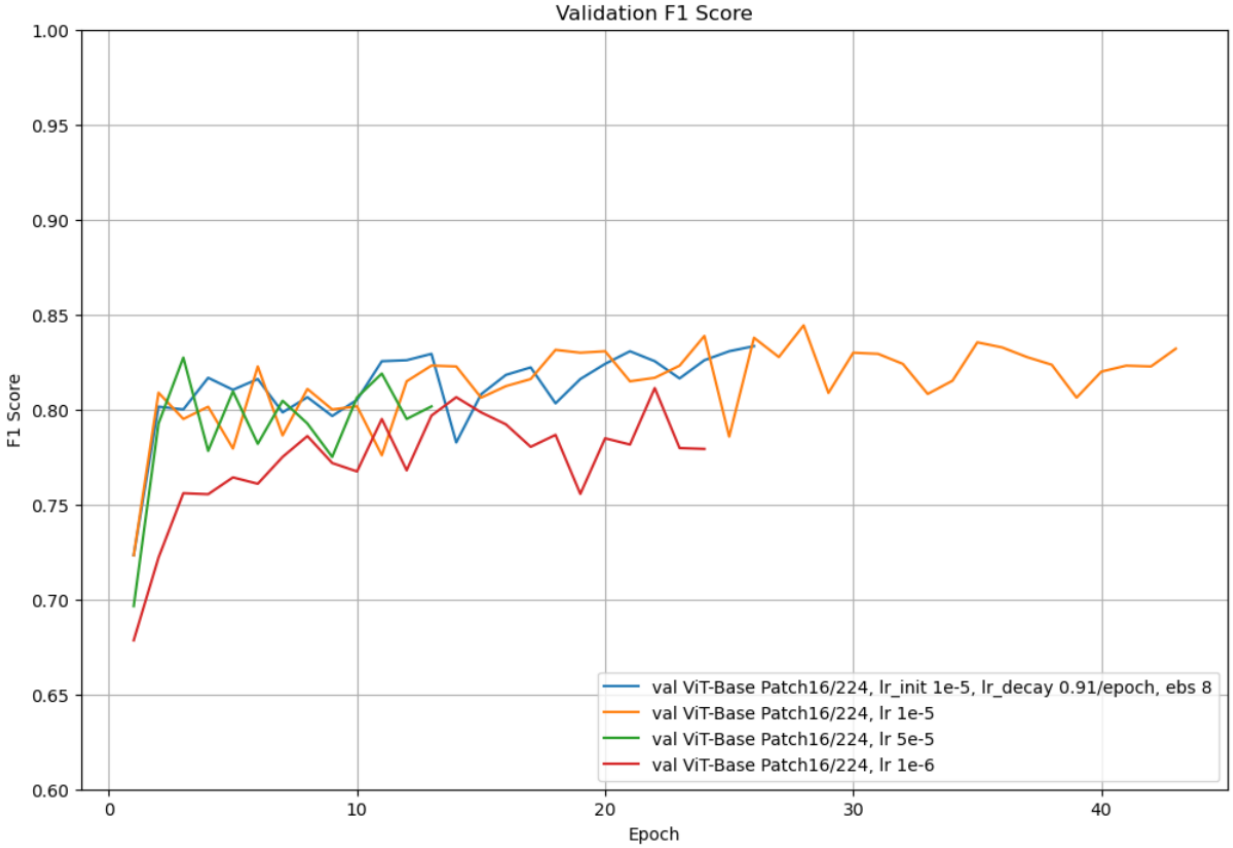
A static rate of 1e-5 gives the best validation F1 score, but since this occurs during non-convergent training behavior, I don't trust this result.

Training results:

Model	Best Epoch	Train Loss	Val Loss	Train AUC	Val AUC	Train F1	Val F1
ViT-Base Patch16/224, lr_init 1e-5, lr_decay 0.91/epoch, ebs 8	11	5.1	30.1	NaN	NaN	0.993	0.825
ViT-Base Patch16/224, lr 1e-5	28	1.9	32.5	NaN	NaN	0.997	0.844
ViT-Base Patch16/224, lr 5e-5	3	49.2	15.4	NaN	NaN	0.932	0.827
ViT-Base Patch16/224, lr 1e-6	14	10.1	23.5	NaN	NaN	0.990	0.806




Training F1 Score
F1 Score
Epoch
train ViT-Base Patch16/224, lr_init 1e-5, lr_decay 0.91/epoch, ebs 8
train ViT-Base Patch16/224, lr 1e-5
train ViT-Base Patch16/224, lr 5e-5
train ViT-Base Patch16/224, lr 1e-6


Validation F1 Score
F1 Score
Epoch
val ViT-Base Patch16/224, lr_init 1e-5, lr_decay 0.91/epoch, ebs 8
val ViT-Base Patch16/224, lr 1e-5
val ViT-Base Patch16/224, lr 5e-5
val ViT-Base Patch16/224, lr 1e-6

3.3 Dropout

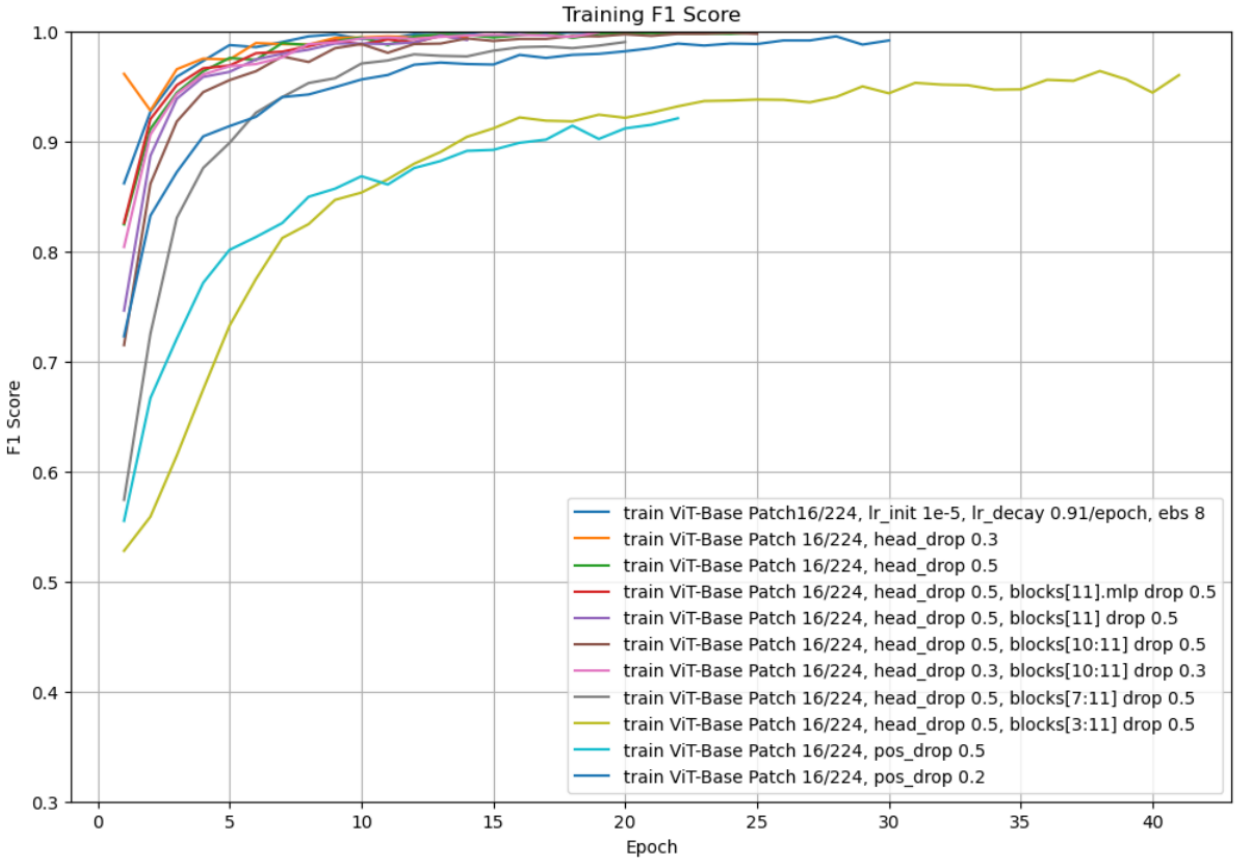
I tried a few different dropout configurations, starting from the final layer and then starting from the first layer.

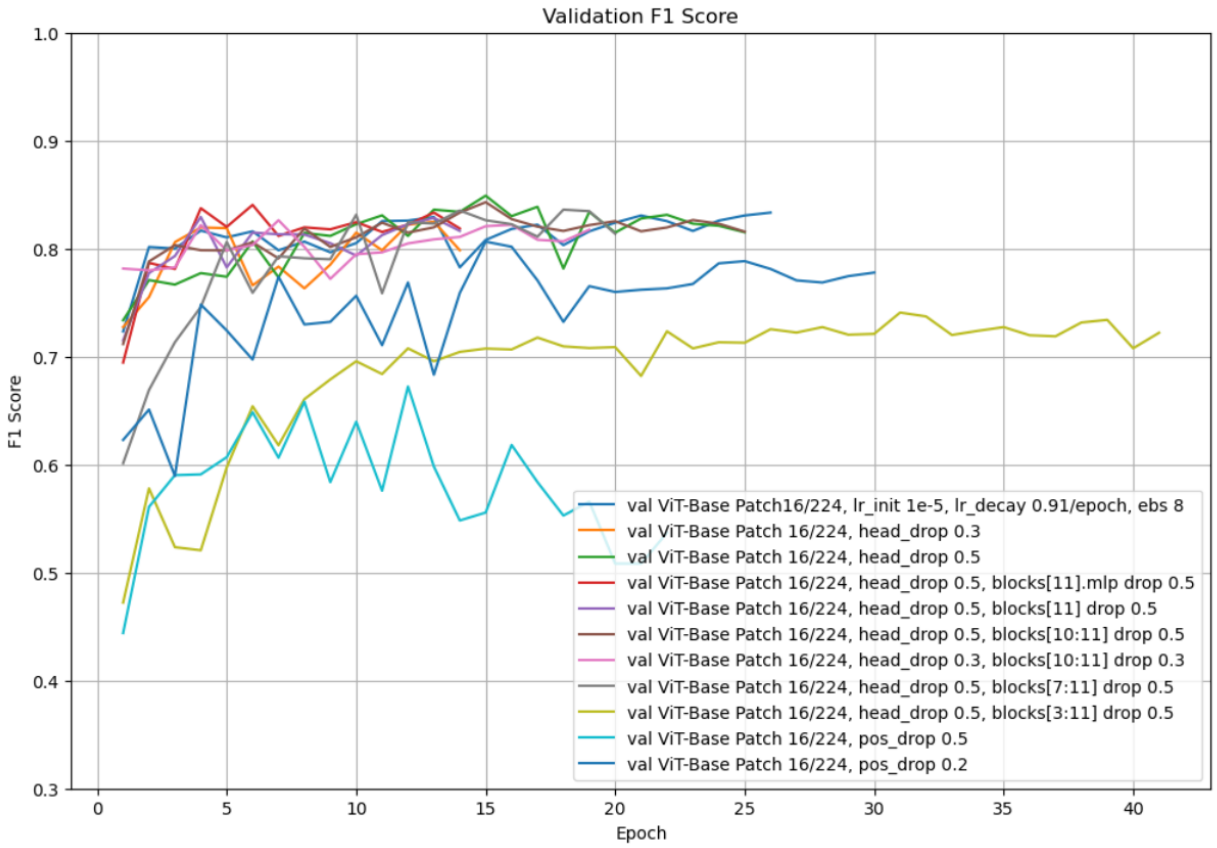
Adding dropout to the first layer adds significant bias to the model without improving generalization. Starting from the final layer and moving back gradually adds bias. Generalization improves a little, but as I added more layers, bias became too high.

Adding dropout rate of 0.5 to only the last layer gave the best validation F1 score.

Training results:

Model	Best Epoch	Train Loss	Val Loss	Train AUC	Val AUC	Train F1	Val F1
ViT-Base Patch16/224, lr_init 1e-5, lr_decay 0.91/epoch, ebs 8	11	5.1	30.1	NaN	NaN	0.993	0.825
ViT-Base Patch 16/224, head_drop 0.3	4	16.8	22.7	NaN	NaN	0.975	0.820
ViT-Base Patch 16/224, head_drop 0.5	15	4.0	31.5	NaN	NaN	0.994	0.849
ViT-Base Patch 16/224, head_drop 0.5, blocks[11].mlp drop 0.5	4	26.7	16.7	NaN	NaN	0.966	0.837
ViT-Base Patch 16/224, head_drop 0.5, blocks[11] drop 0.5	4	30.0	20.5	NaN	NaN	0.958	0.829
ViT-Base Patch 16/224, head_drop 0.5, blocks[10:11] drop 0.5	15	6.2	39.0	NaN	NaN	0.991	0.843
ViT-Base Patch 16/224, head_drop 0.3, blocks[10:11] drop 0.3	4	29.0	23.5	NaN	NaN	0.961	0.822
ViT-Base Patch 16/224, head_drop 0.5, blocks[7:11] drop 0.5	10	23.4	35.6	NaN	NaN	0.971	0.831
ViT-Base Patch 16/224, head_drop 0.5, blocks[3:11] drop 0.5	31	35.2	94.2	NaN	NaN	0.953	0.741
ViT-Base Patch 16/224, pos_drop 0.5	12	74.8	27.7	NaN	NaN	0.876	0.672
ViT-Base Patch 16/224, pos_drop 0.2	15	20.3	27.2	NaN	NaN	0.970	0.806


Training F1 Score
F1 Score
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
5
10
15
20
25
30
35
40
train ViT-Base Patch16/224, lr_init 1e-5, lr_decay 0.91/epoch, ebs 8
train ViT-Base Patch 16/224, head_drop 0.3
train ViT-Base Patch 16/224, head_drop 0.5
train ViT-Base Patch 16/224, head_drop 0.5, blocks[11].mlp drop 0.5
train ViT-Base Patch 16/224, head_drop 0.5, blocks[11] drop 0.5
train ViT-Base Patch 16/224, head_drop 0.5, blocks[10:11] drop 0.5
train ViT-Base Patch 16/224, head_drop 0.3, blocks[10:11] drop 0.3
train ViT-Base Patch 16/224, head_drop 0.5, blocks[7:11] drop 0.5
train ViT-Base Patch 16/224, head_drop 0.5, blocks[3:11] drop 0.5
train ViT-Base Patch 16/224, pos_drop 0.5
train ViT-Base Patch 16/224, pos_drop 0.2



3.4 Weight decay

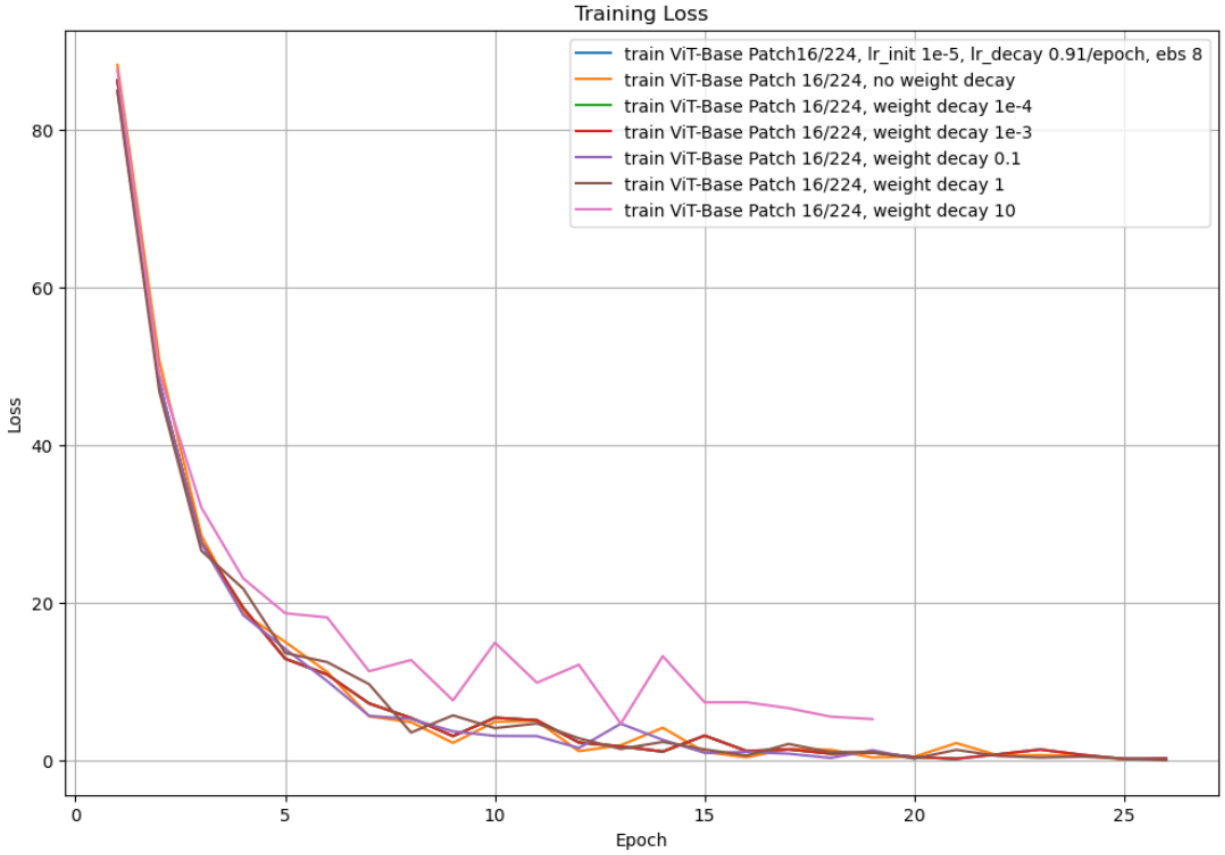
For these experiments, I switched to the AdamW optimizer since it's supposed to give more stable results with weight decay.

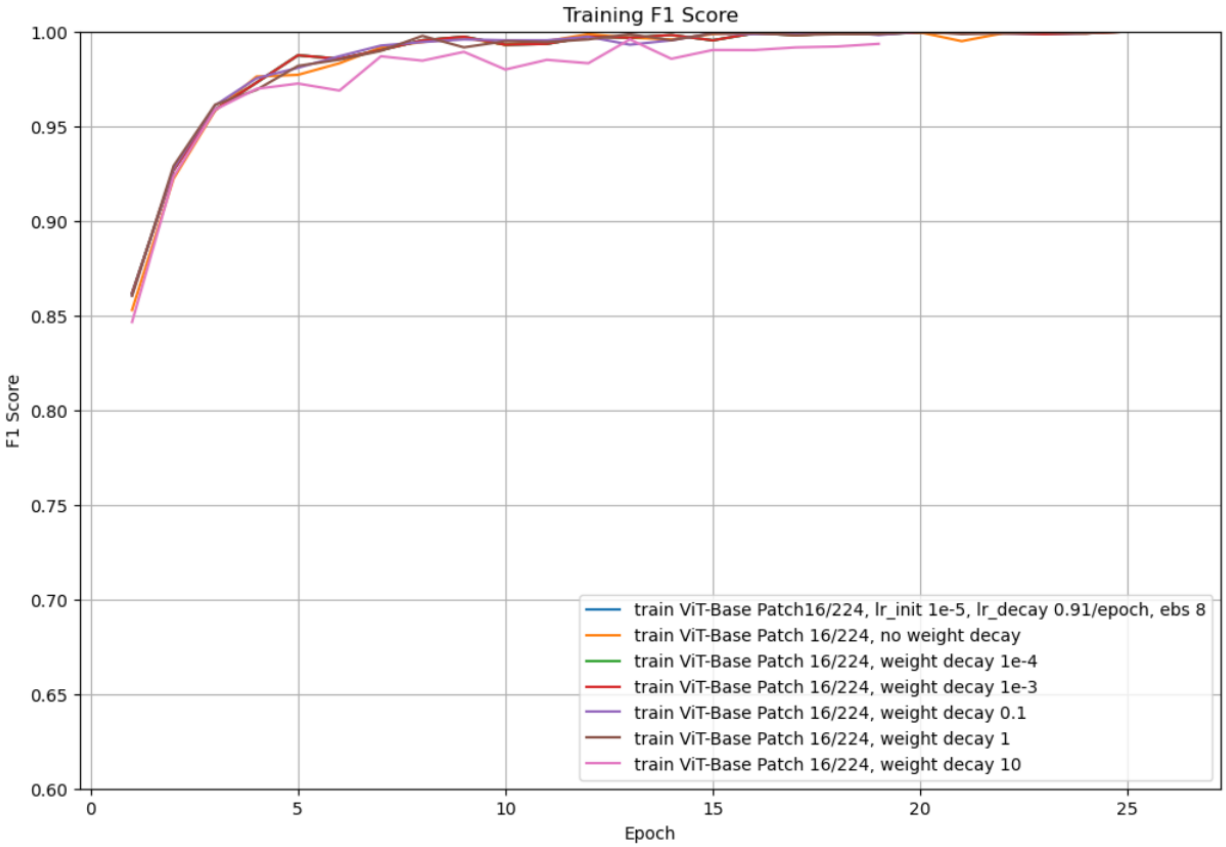
As a sanity-check, I trained with the AdamW optimizer without weight decay to compare against the Adam optimizer. The results don't match, but I didn't have time to investigate.

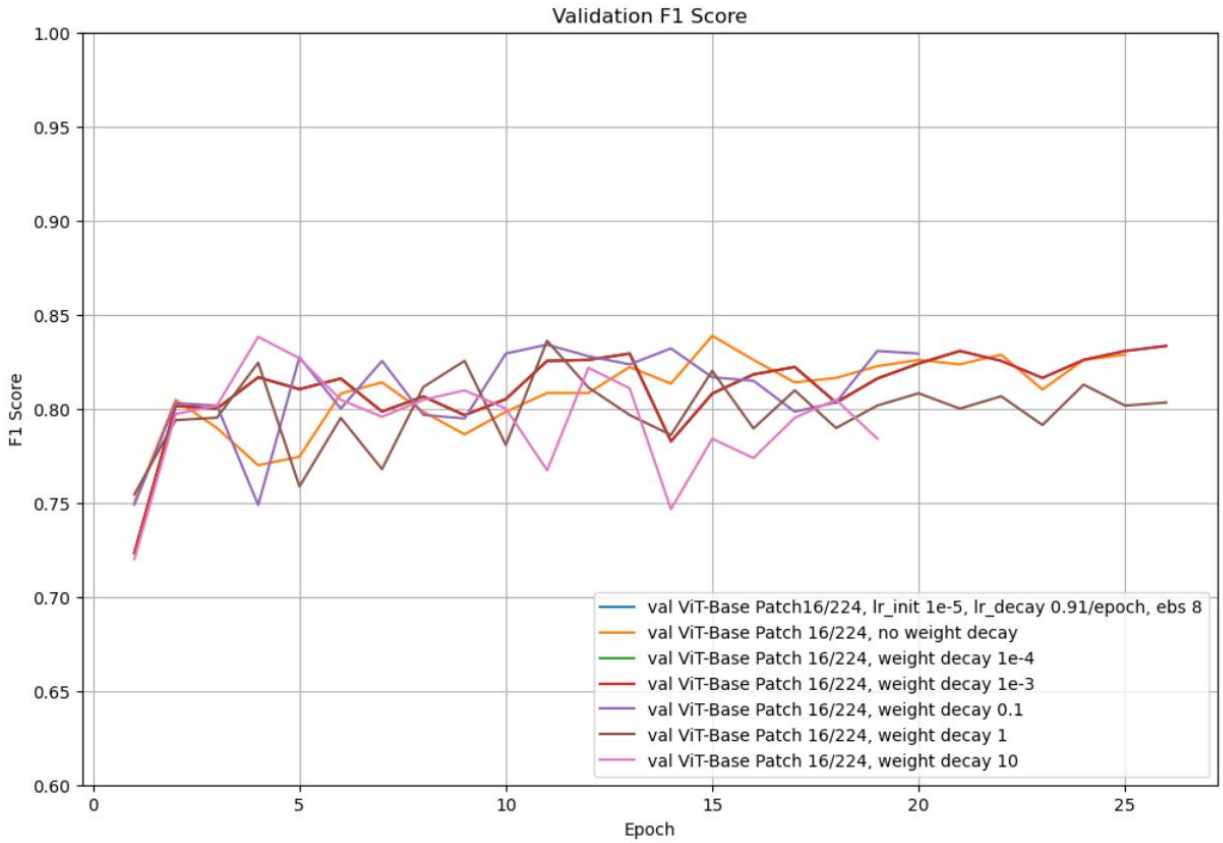
However, the results do match for small weight decays, and higher weight decay does seem to improve validation F1 score. It looks like learning rate needs to be adjusted for a weight decay of 10. I didn't have time to try combinations of dropout and weight decay, so I went with dropout only.

Training results:

Model	Best Epoch	Train Loss	Val Loss	Train AUC	Val AUC	Train F1	Val F1
ViT-Base Patch16/224, lr_init 1e-5, lr_decay 0.91/epoch, ebs 8	11	5.1	30.1	NaN	NaN	0.993	0.825
ViT-Base Patch 16/224, no weight decay	15	1.1	32.0	NaN	NaN	0.999	0.839
ViT-Base Patch 16/224, weight decay 1e-4	11	5.1	30.1	NaN	NaN	0.993	0.825
ViT-Base Patch 16/224, weight decay 1e-3	11	5.1	30.1	NaN	NaN	0.993	0.825
ViT-Base Patch 16/224, weight decay 0.1	5	14.2	26.4	NaN	NaN	0.981	0.827
ViT-Base Patch 16/224, weight decay 1	11	4.7	31.1	NaN	NaN	0.994	0.836
ViT-Base Patch 16/224, weight decay 10	4	23.1	22.4	NaN	NaN	0.970	0.838




Training F1 Score
F1 Score
Epoch
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
5
10
15
20
25
train ViT-Base Patch16/224, lr_init 1e-5, lr_decay 0.91/epoch, ebs 8
train ViT-Base Patch 16/224, no weight decay
train ViT-Base Patch 16/224, weight decay 1e-4
train ViT-Base Patch 16/224, weight decay 1e-3
train ViT-Base Patch 16/224, weight decay 0.1
train ViT-Base Patch 16/224, weight decay 1
train ViT-Base Patch 16/224, weight decay 10

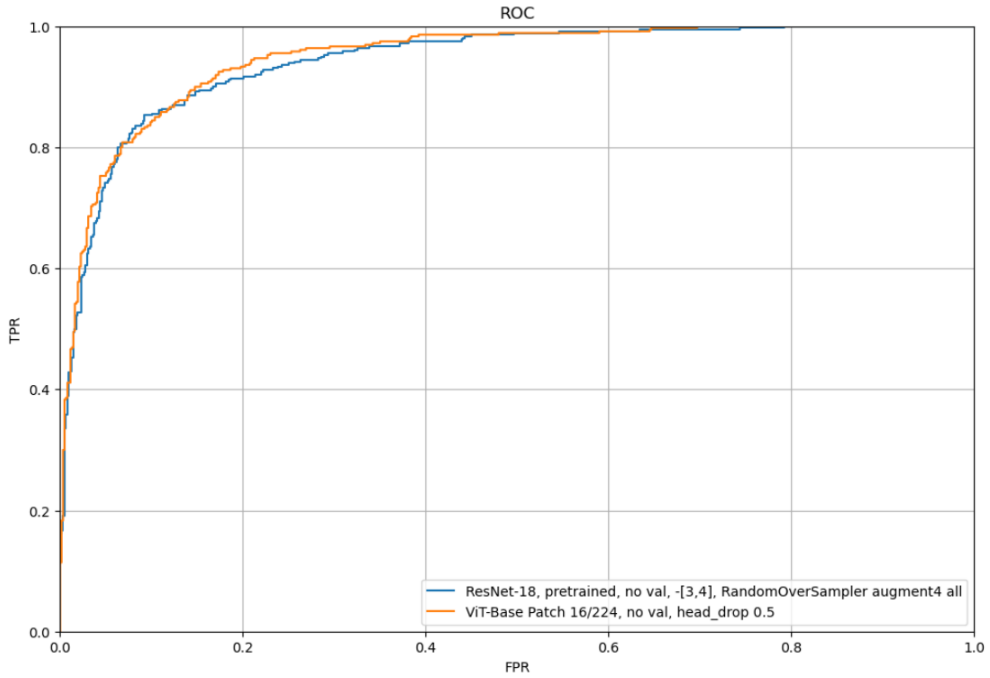


3.5 ResNet-18 vs. ViT-Base Patch 16/224

For a single training/testing run, the best performance for ResNet and ViT are comparable.

Testing results:

Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
ResNet-18, pretrained, no val, -[3,4], RandomOverSampler augment4 all	13.717	0.861	0.825	0.922	0.843	0.886	0.874	0.942
ViT-Base Patch 16/224, no val, head_drop 0.5	73.056	0.832	0.842	0.901	0.837	0.879	0.871	0.948

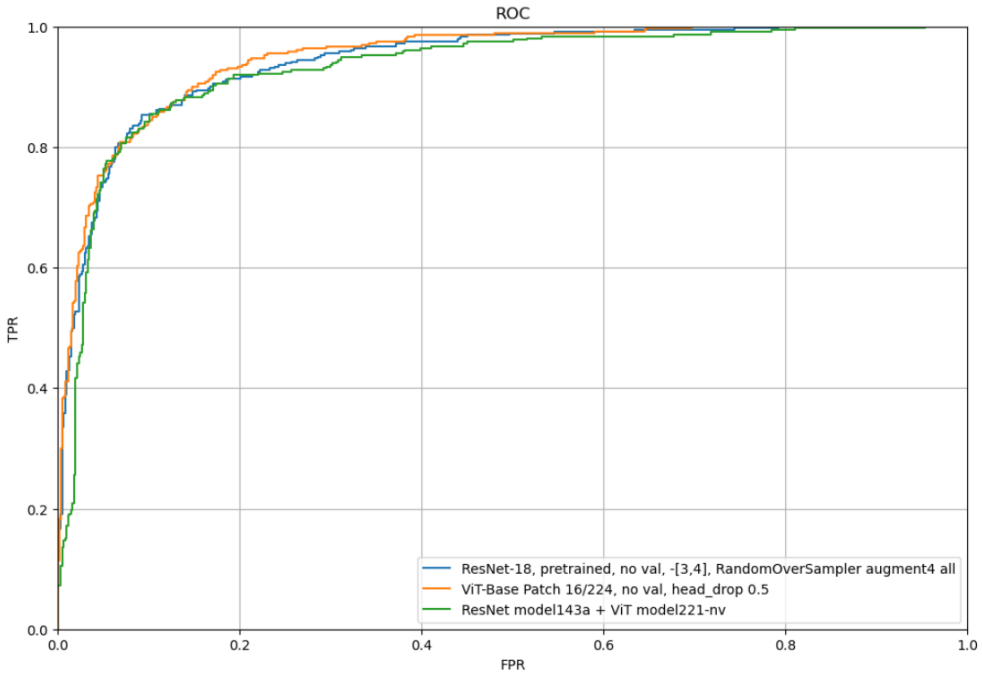


4 Ensembling

Ensembling the best ResNet and ViT models results in worse performance than either of them, but I didn't have time to investigate.

Testing results:

Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
ResNet-18, pretrained, no val, -[3,4], RandomOverSampler augment4 all	13.717	0.861	0.825	0.922	0.843	0.886	0.874	0.942
ViT-Base Patch 16/224, no val, head_drop 0.5	73.056	0.832	0.842	0.901	0.837	0.879	0.871	0.948
ResNet model143a + ViT model221-nv	52.597	0.881	0.783	0.938	0.829	0.881	0.861	0.931

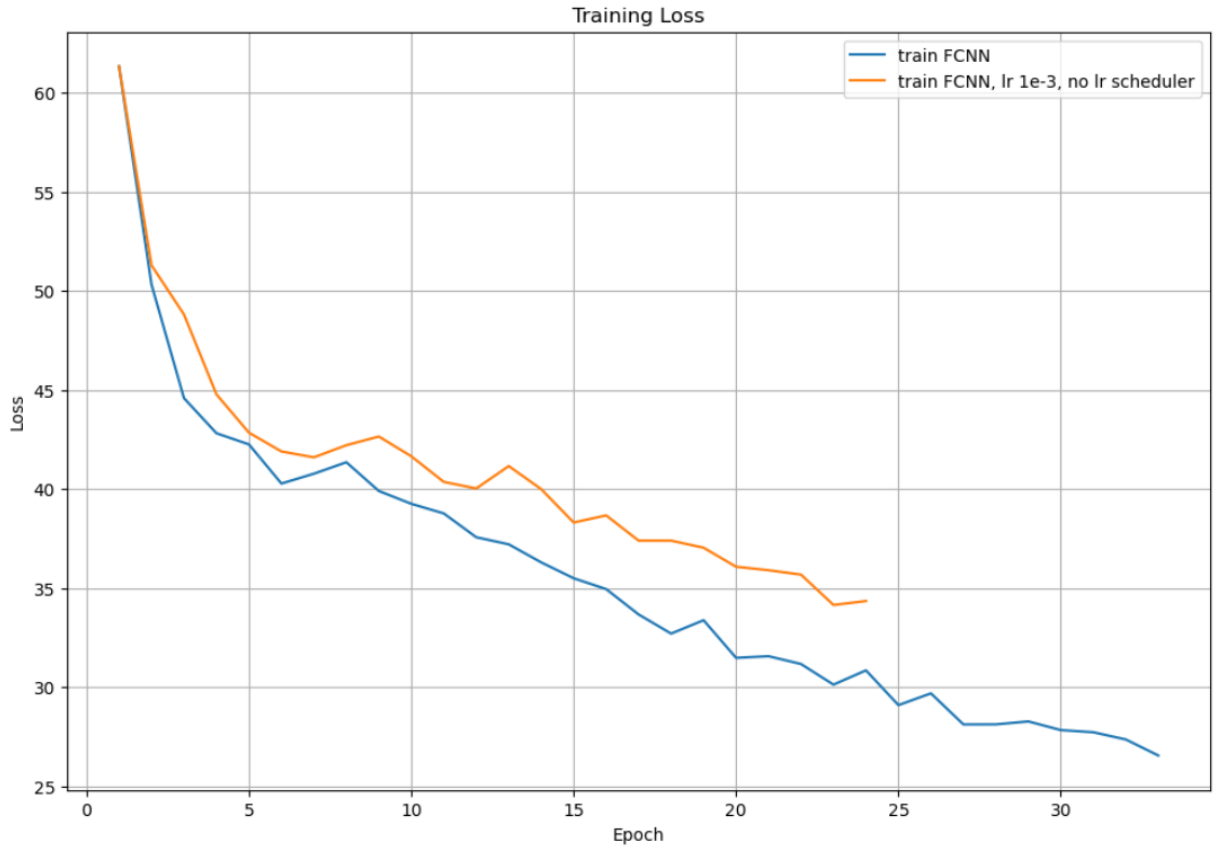


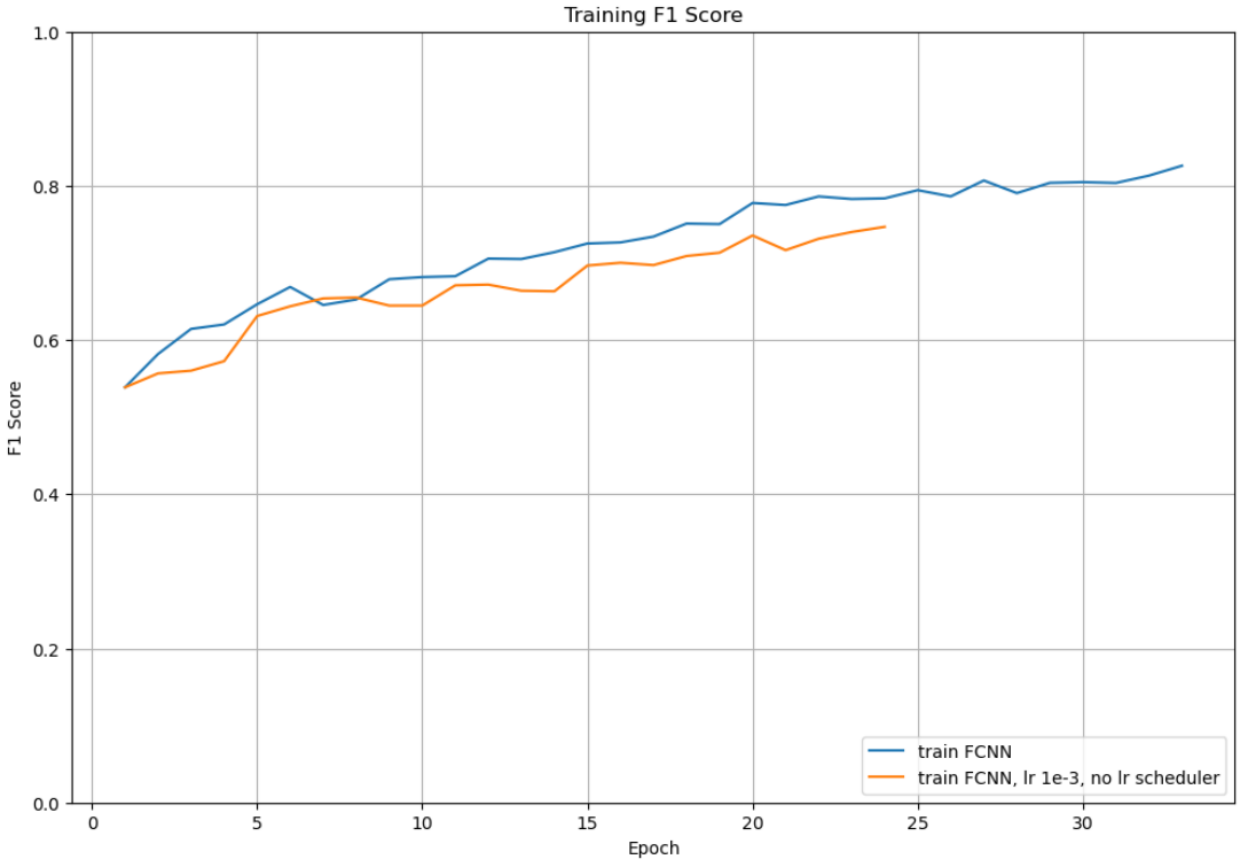
5 FCNN

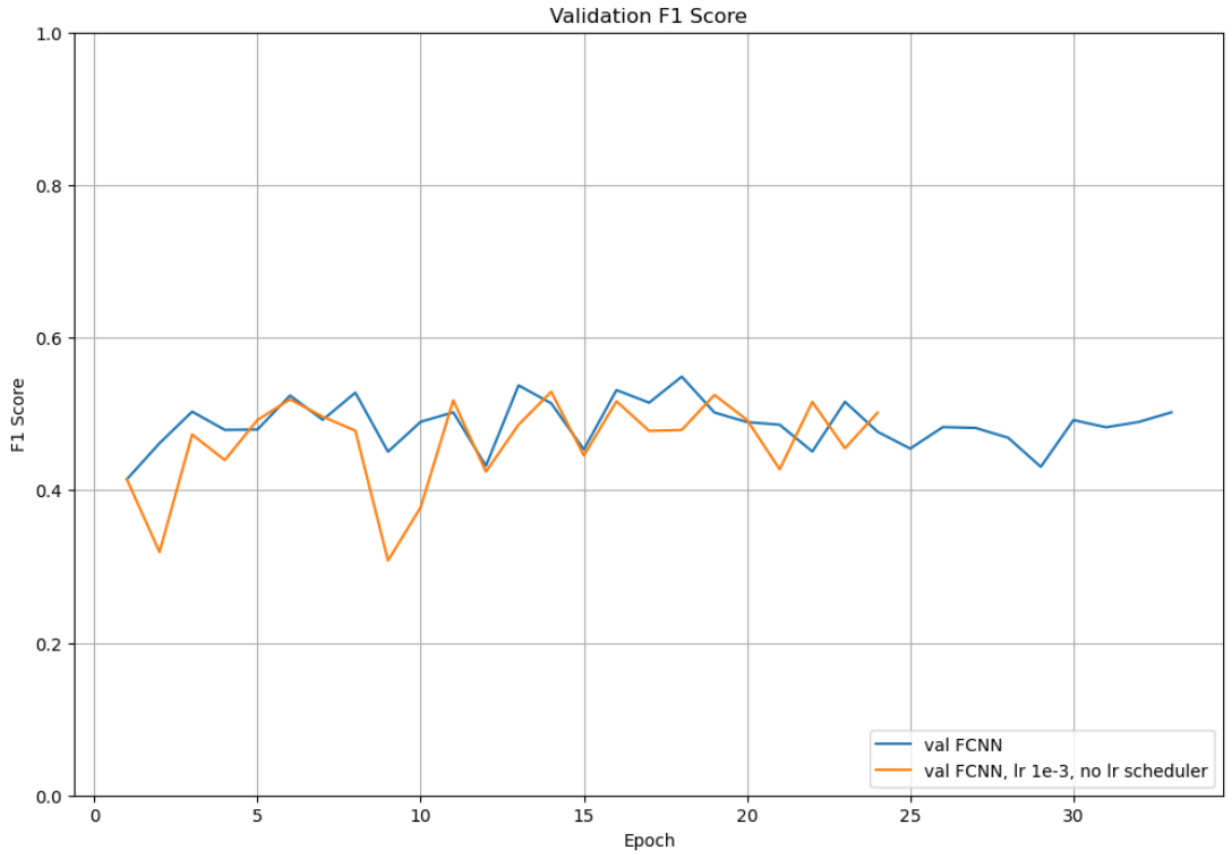
For the FCNN, I used the same hyperparameters as the best ResNet-18 configuration. The only difference is that I used mean and standard deviation of 0.5 for normalizing the pixel values to [-1, 1] in preprocessing. I only ran one experiment, removing the LR scheduler.

Training results:

Model	Best Epoch	Train Loss	Val Loss	Train AUC	Val AUC	Train F1	Val F1
FCNN	18	32.7	9.1	NaN	NaN	0.751	0.549
FCNN, lr 1e-3, no lr scheduler	14	40.0	9.0	NaN	NaN	0.663	0.529

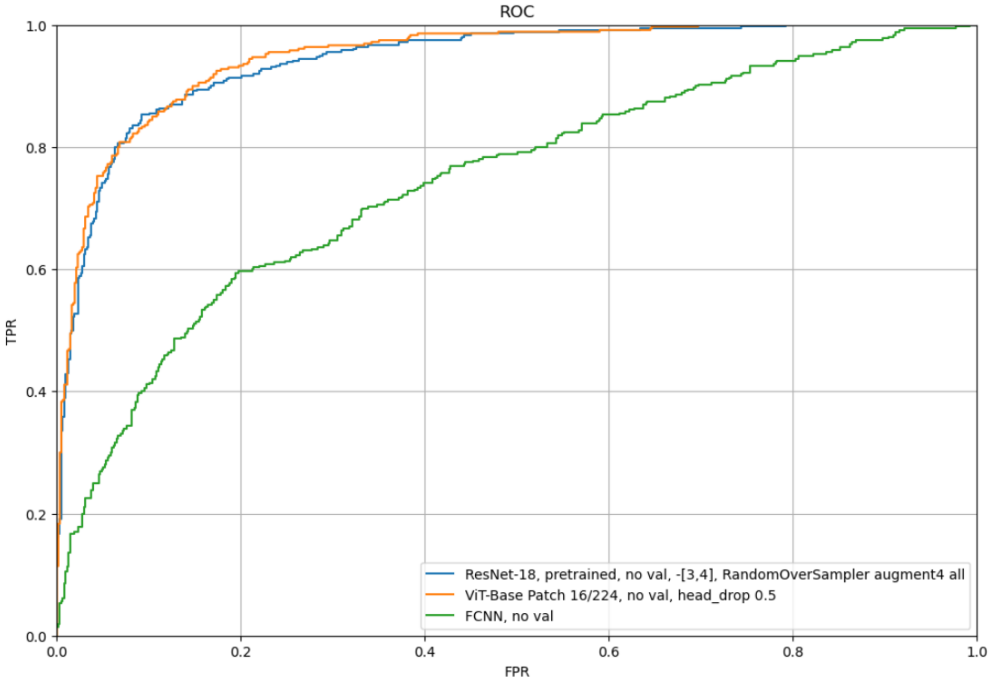



Training F1 Score
F1 Score
Epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
train FCNN
train FCNN, lr 1e-3, no lr scheduler



Testing results:

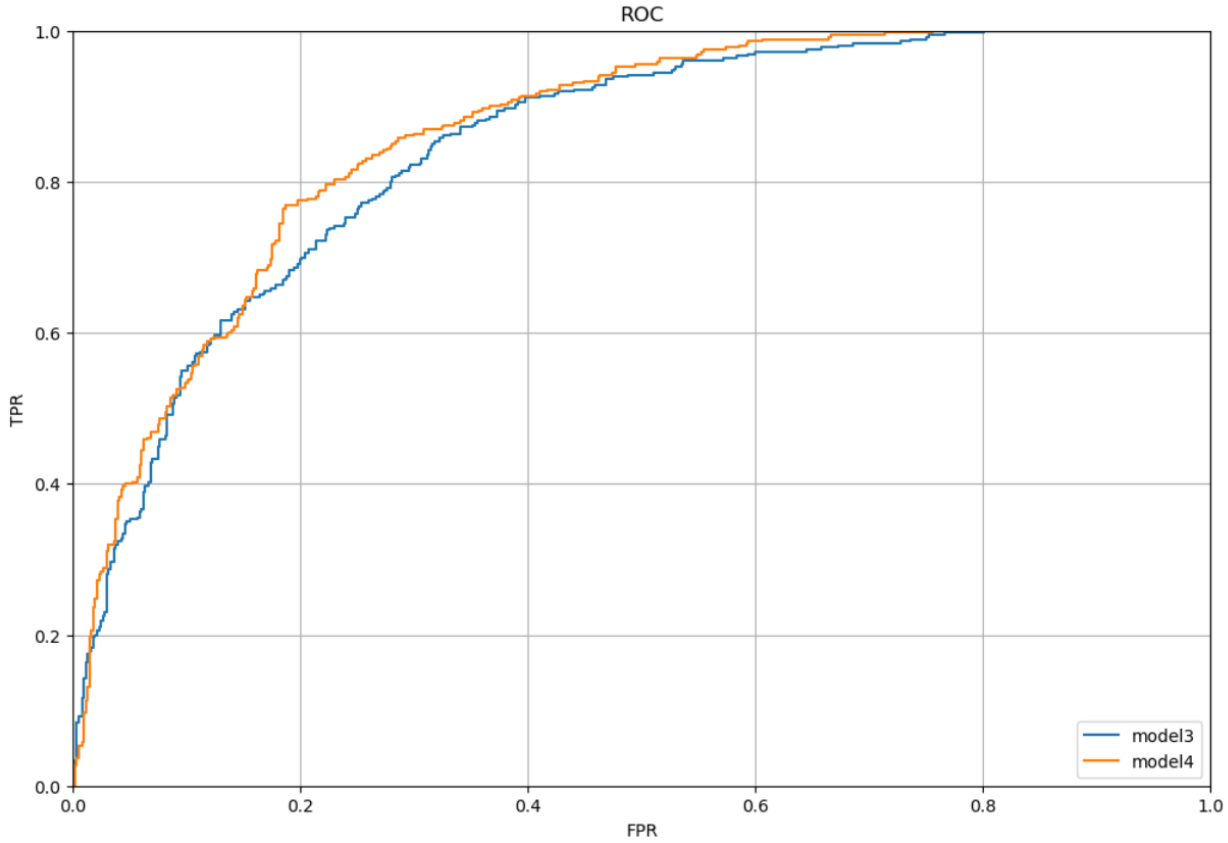
Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
ResNet-18, pretrained, no val, -[3,4], RandomOverSampler augment4 all	13.717	0.861	0.825	0.922	0.843	0.886	0.874	0.942
ViT-Base Patch 16/224, no val, head_drop 0.5	73.056	0.832	0.842	0.901	0.837	0.879	0.871	0.948
FCNN, no val	19.160	0.641	0.581	0.810	0.609	0.726	0.695	0.743



6 Appendix

6.1 Torchmetrics AUROC problem

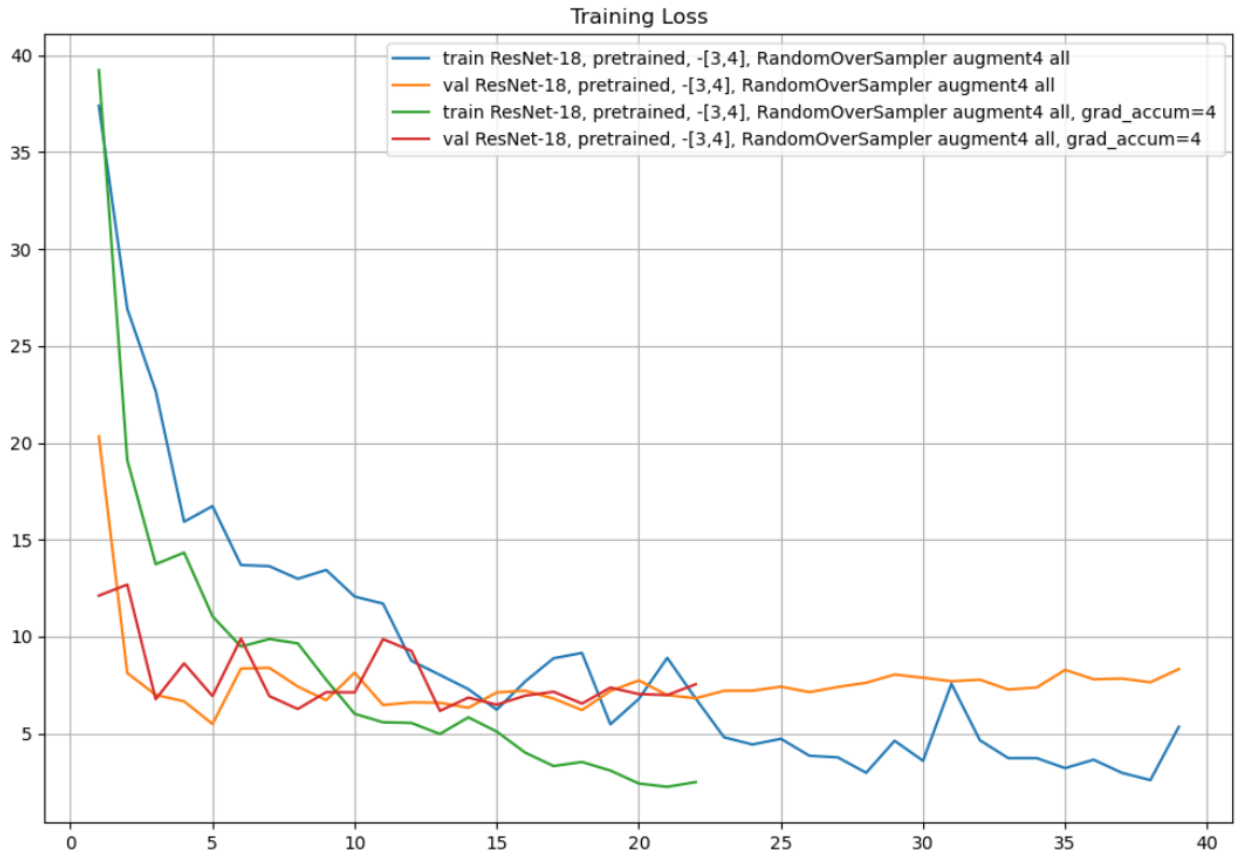
Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
model3	17.971	0.671	0.692	0.802	0.681	0.762	0.747	0.845
model4	16.246	0.777	0.514	0.914	0.619	0.767	0.714	0.861



6.2 Gradient accumulation with ResNet

With grad_accum=4 (accumulate 4 batches of size 32 per update), testing performance doesn't change, but training converges much more quickly.

Training data:



Testing data:

Model	Loss	BinaryPrecision	BinaryRecall	BinarySpecificity	BinaryF1Score	BinaryAccuracy	BinaryAUROC	AUC Manual Calc
ResNet-18, pretrained, -[3,4], RandomOverSampler augment4 all	14.074	0.815	0.842	0.888	0.828	0.871	0.865	0.937
ResNet-18, pretrained, -[3,4], RandomOverSampler augment4 all, grad_accum=4	14.127	0.848	0.822	0.914	0.835	0.880	0.868	0.935