EE 443 Capstone Presentation

Jonathan Wong

Dataset	Baseline	?	?	?	?	?
Full						
0.1						
0.02						
0.005						
	Average Improvement:					

Welcome!

EE 443 Capstone Presentation

Jonathan Wong

Dataset	Baseline	?	?	?	?	?
Full						
0.1						
0.02						
0.005						
Average Improvement:						

Experiment #1: Establishing a Baseline

Motivation:

Establish point of reference for evaluating performance of future experiments.

Procedures:

Architecture Sweep, Hyperparameter Sweep.

Establishing a Baseline: Code

Dataset

```python
class BaseDataset(Dataset):
  def __init__(self, datastore, input, image_size=224, da=False):
    if isinstance(input, str):
      with open(input) as f:
        json_data = json.load(f)
        self.data = json_data['annotations']
    elif isinstance(input, list):
      self.data = input

    self.datastore = datastore
    self.image_size = image_size
    self.da = da
    self.transform = A.Compose(
      [
        A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.15, rotate_limit=15, p=0.5),
        A.RGBShift(r_shift_limit=15, g_shift_limit=15, b_shift_limit=15, p=0.5),
        A.RandomBrightnessContrast(p=0.5)
      ]
    )

  def __len__(self):
    return len(self.data)
```

...

Loads Json, Data Augmentation, Normalization

DataLoader

```python
def get_dataloaders(dataset, batch_size=16, num_workers=2):
  # Calculate Split
  val_split = 0.2   # Hardcoded 20%
  dataset_size = len(dataset)
  indices = list(range(dataset_size))
  split = int(np.floor(val_split * dataset_size))

  # Shuffle Data
  np.random.shuffle(indices)

  # Split Base Dataset into Training and Validation Datasets
  train_indices, val_indices = indices[split:], indices[:split]
  train_sampler = SubsetRandomSampler(train_indices)
  val_sampler = SubsetRandomSampler(val_indices)

  # Dataloaders
  train_dataloader = DataLoader(dataset, batch_size, num_workers=num_workers, sampler=train_sampler)
  val_dataloader = DataLoader(dataset, batch_size, num_workers=num_workers, sampler=val_sampler)

  return train_dataloader, val_dataloader

def get_test_dataloader(datastore, test_json):
  testset = BaseDataset(datastore, test_json)
  test_dataloader = DataLoader(testset, batch_size=32, num_workers=2)
  return test_dataloader
```

...

Sets Batch Size

Establishing a Baseline: Code

Model

```python
class BaseNet(LightningModule):
  def __init__(self, model_type='efficientnet_b0', lr=4e-4, weighted_loss=None, hist=None, beta=None):
    super().__init__()
    self.save_hyperparameters()

    # Create backbone
    backbone = timm.create_model(model_type, pretrained=True)
    layers = list(backbone.children())[:-1]
    [fc] = list(backbone.children())[-1:]
    self.feature_extractor = nn.Sequential(*layers)
    self.classifer = nn.Linear(fc.in_features, 50)

    # Weighted Loss
    self.weighted_loss = WeightedLoss(weighted_loss, hist, beta)

    # Metrics
    self.train_acc = torchmetrics.Accuracy()
    self.valid_acc = torchmetrics.Accuracy()
    self.test_acc = torchmetrics.Accuracy()
    self.preds = []
    self.labels = []
```

...

Architecture, Metrics, Loss Function

Training Script

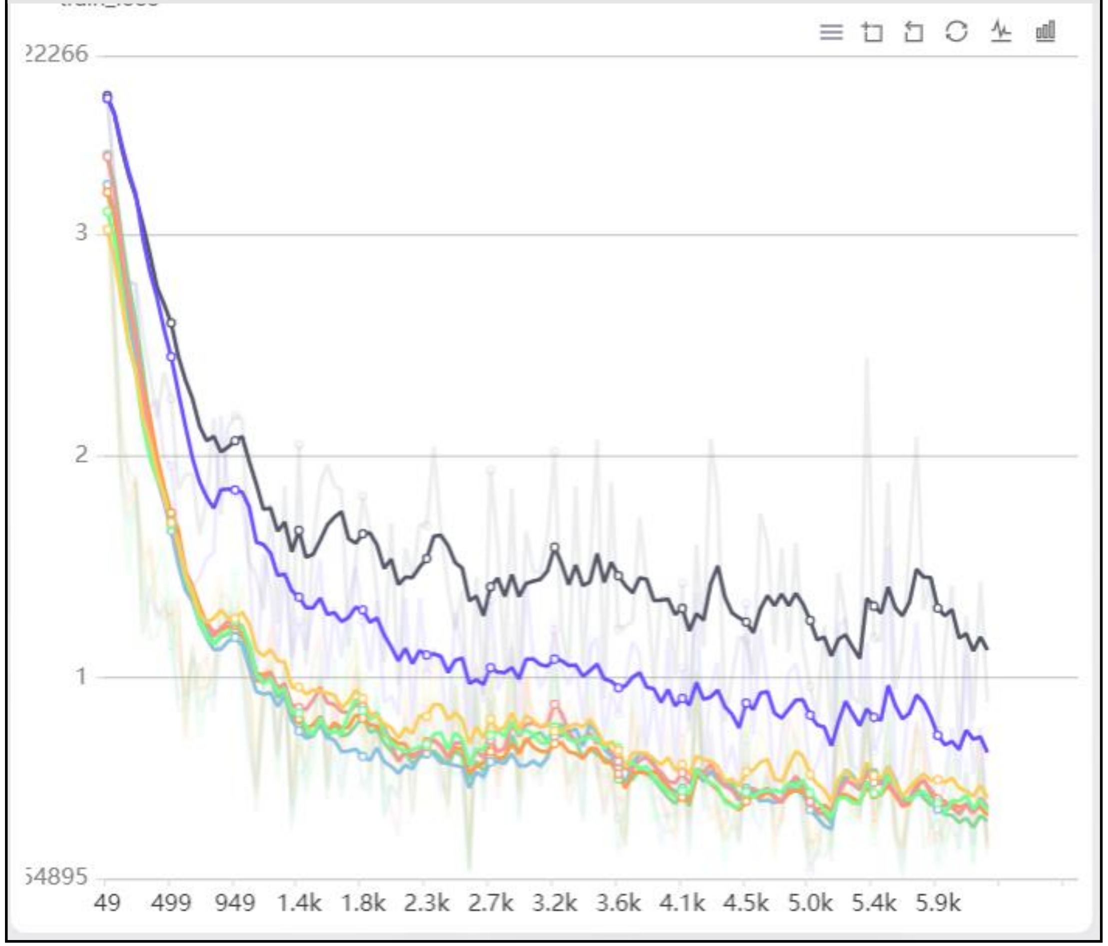
```python
class Baseline(ExperimentInterface):
  def run_experiment(self, args: dict) -> None:
    # 1) Init Data Components
    train_json = str(Path(args['datastore']) / 'train.json')
    test_json = str(Path(args['datastore']) / 'test.json')
    dataset = BaseDataset(args['datastore'], train_json, image_size=args['image_size'], da=args['augment_data'])

    train_dataloader, val_dataloader = get_dataloaders(dataset, batch_size=args['batch_size'], num_workers=args['num_workers'])
    test_dataloader = get_test_dataloader(args['datastore'], test_json)

    # 2) Init Model
    if args['weighted_loss'] is not None:
      hist = get_histogram(process_json(train_json))
      model = BaseNet(model_type=args['model'], lr=args['lr'], weighted_loss=args['weighted_loss'], hist=hist, beta=args['beta'])
    else:
      model = BaseNet(model_type=args['model'], lr=args['lr'])

    # 3) Init Trainer
    trainer = Trainer(gpus=args['gpus'], max_epochs=args['epochs'],
                      checkpoint_callback=True,
                      logger=TensorBoardLogger(save_dir='lightning_logs'))

    # 4) Run Training
    trainer.fit(model, train_dataloader, val_dataloader)
    trainer.save_checkpoint("training_end.ckpt")

    # 5) Run Inference
    result = trainer.test(model, test_dataloader)
    print(result)
```

...

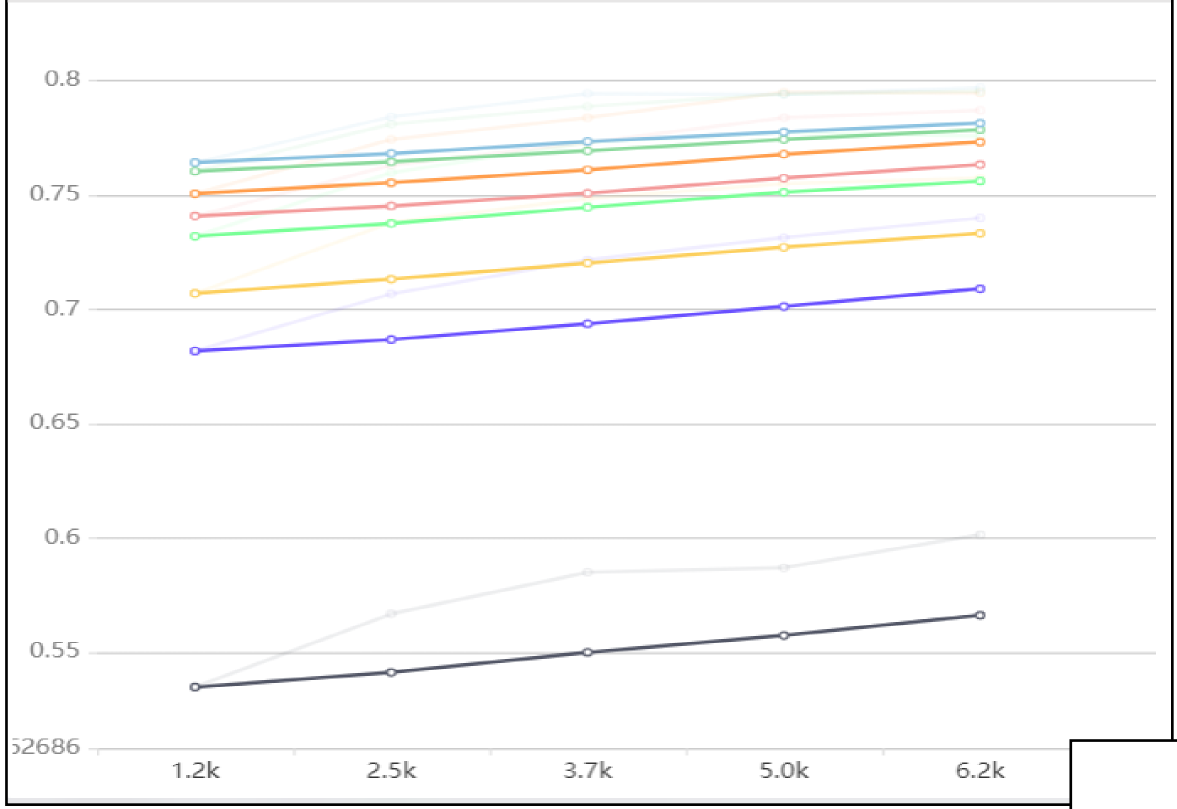
Dataset, DataLoader, Trainer

Establishing a Baseline: Architecture Sweep

Training Loss



Validation Accuracy




```
train.py \
--datastore grid:cifar50:2 \
--model "['resnet34d', 'resnet50', 'resnet50d',
'densenet121', 'efficientnet_b0', 'efficientnet_b1_pruned']"
\
```

GRID.AI

Establishing a Baseline: Architecture Sweep

Results

Model Name	Test Accuracy	# Parameters
efficientnet_b2	0.804	10 M
resnet50d	0.793	22 M
efficientnet_b1	0.7892	8 M
efficientnet_b0	0.788	5 M
densenet121	0.770	20 M
resnet50	0.758	22 M
resnet34d	0.730	20 M

Selected **efficientnet0** architecture as baseline.

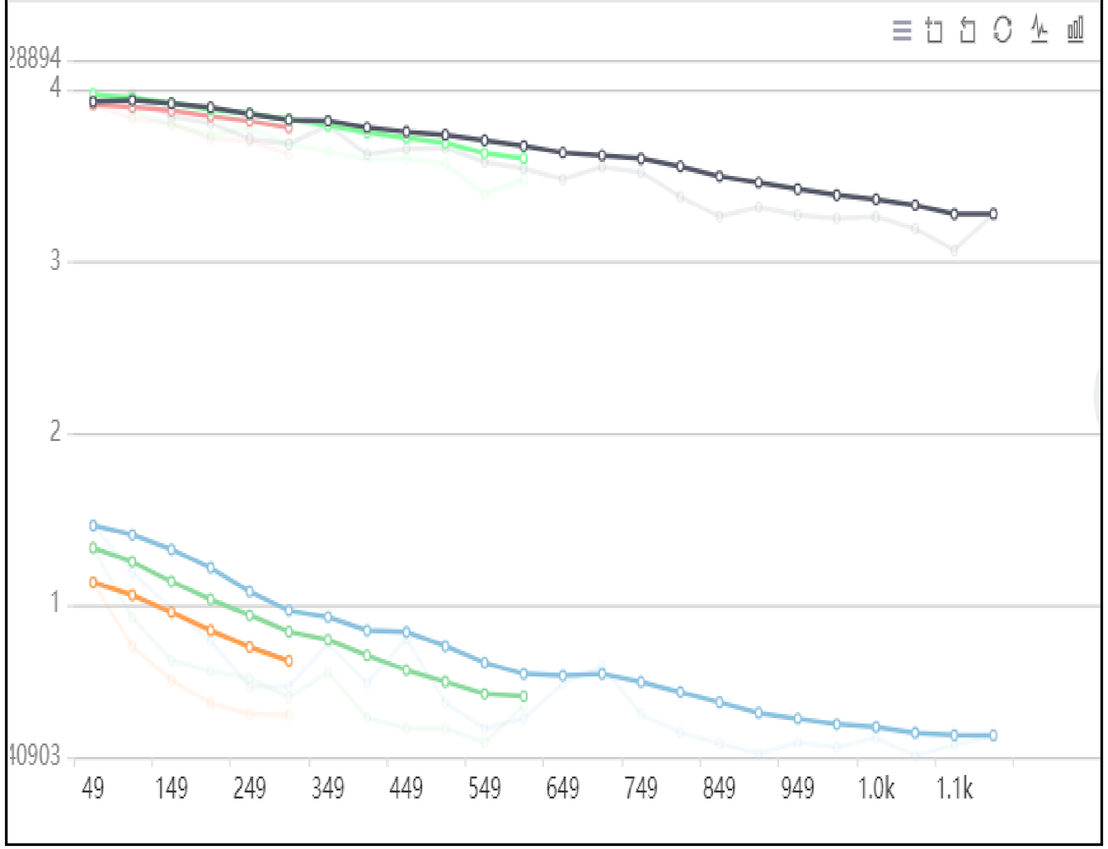
Least parameters for highest accuracy for quick training times.

Analysis:

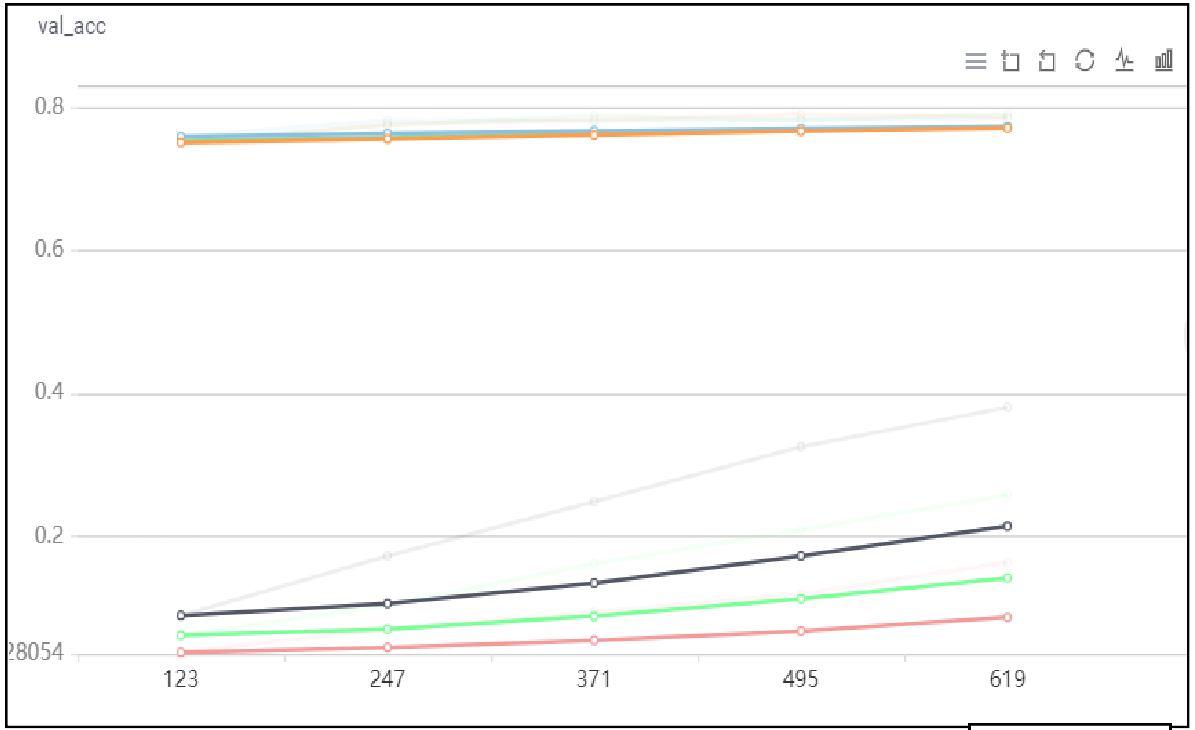
- Models with more hyperparameters tend to overfit
- EfficientNet family is most accurate for the least parameters, hence its name!

Establishing a Baseline: Hyperparameter Sweep

Training Loss



Validation Accuracy



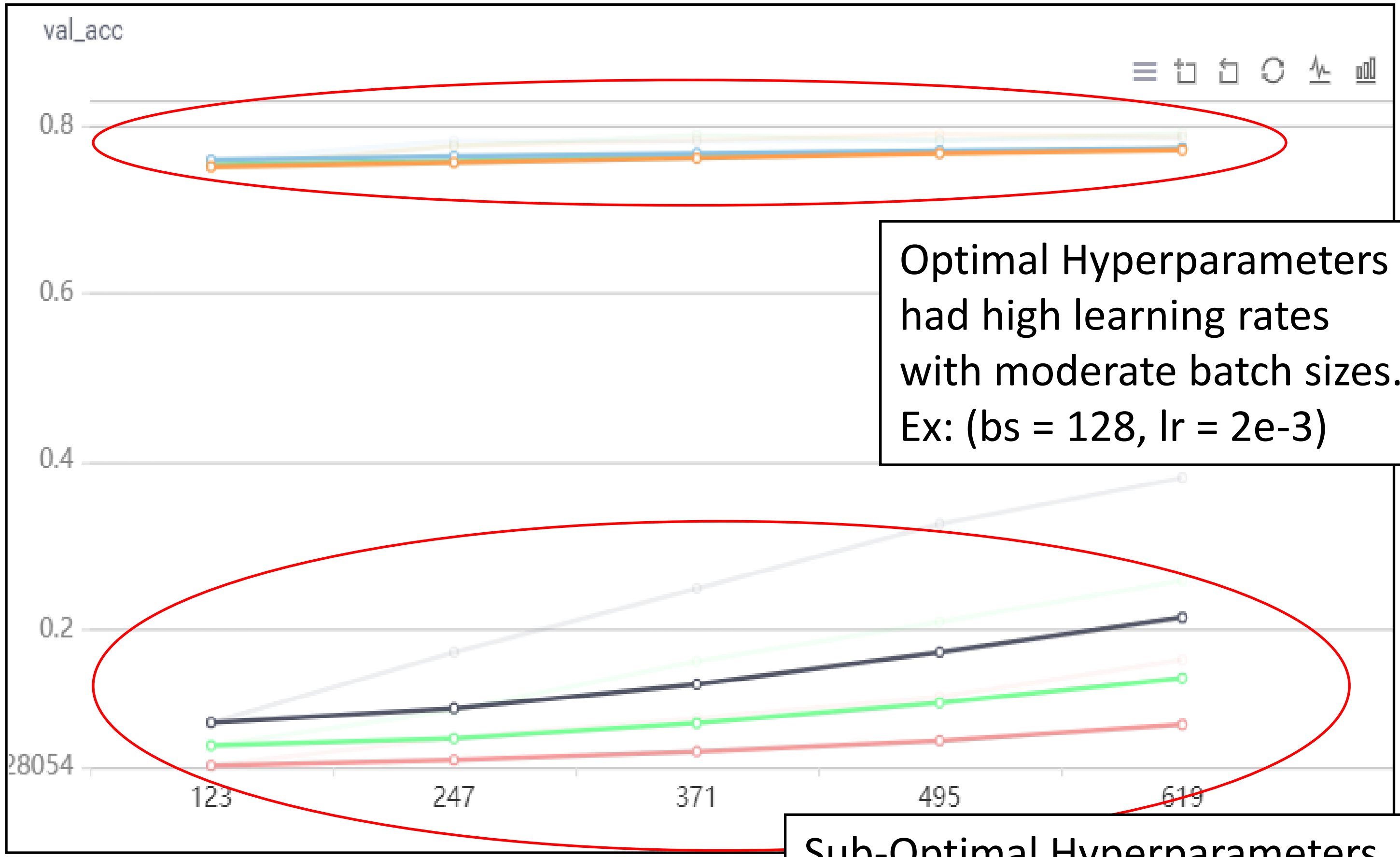

```
train.py \
--datastore grid:cifar50-imbalance-01:2 \
--model efficientnet_b0 \
--batch_size "[16, 32, 64, 128]" \
--lr "[1e-5, 4e-4, 8e-3, 2e-3]"
```

Establishing a Baseline: Hyperparameter Sweep

Results

Repeat sweep for each dataset…

Dataset	Test Accuracy
Full	0.792
0.1	0.7296
0.02	0.6304
0.005	0.5446



Optimal Hyperparameters had high learning rates with moderate batch sizes. Ex: (bs = 128, lr = 2e-3)

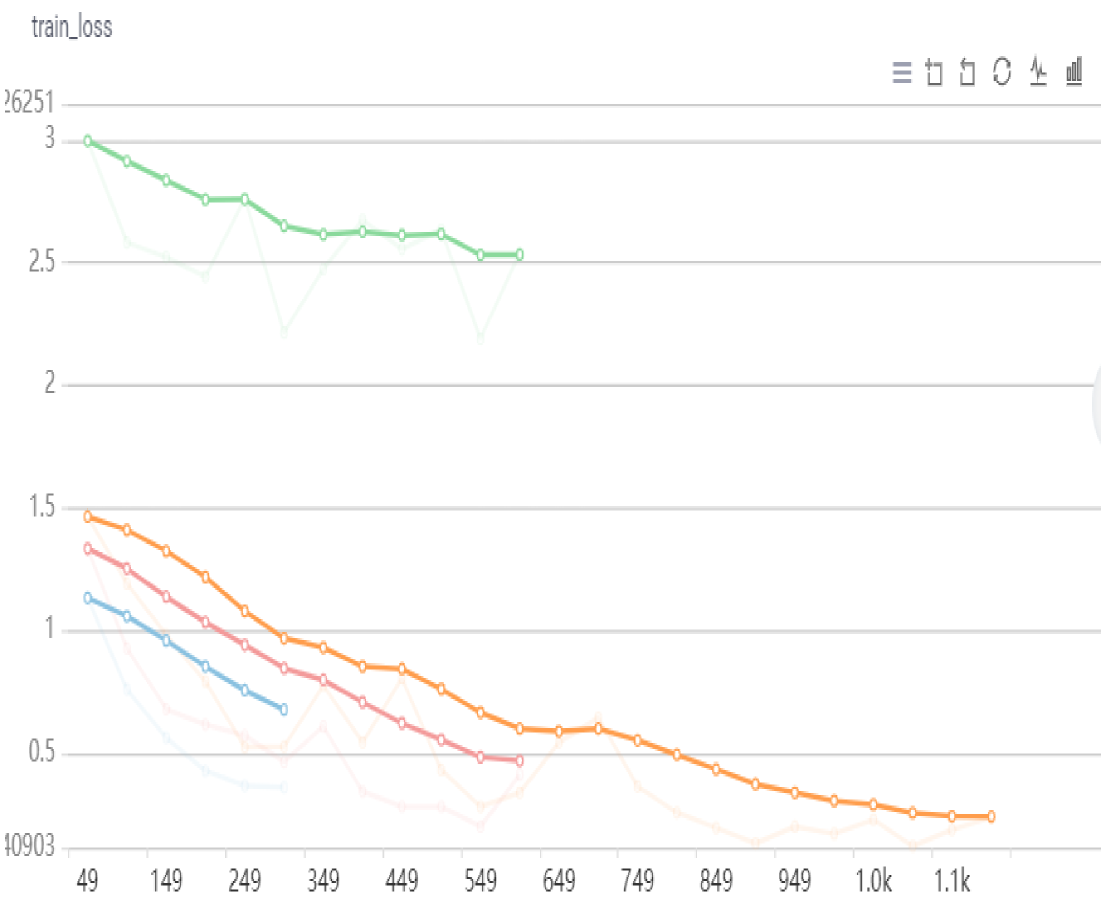
Sub-Optimal Hyperparameters had extremely low learning rates and large batch sizes. Ex: (bs = 512, lr = 1e-5)

Analysis:

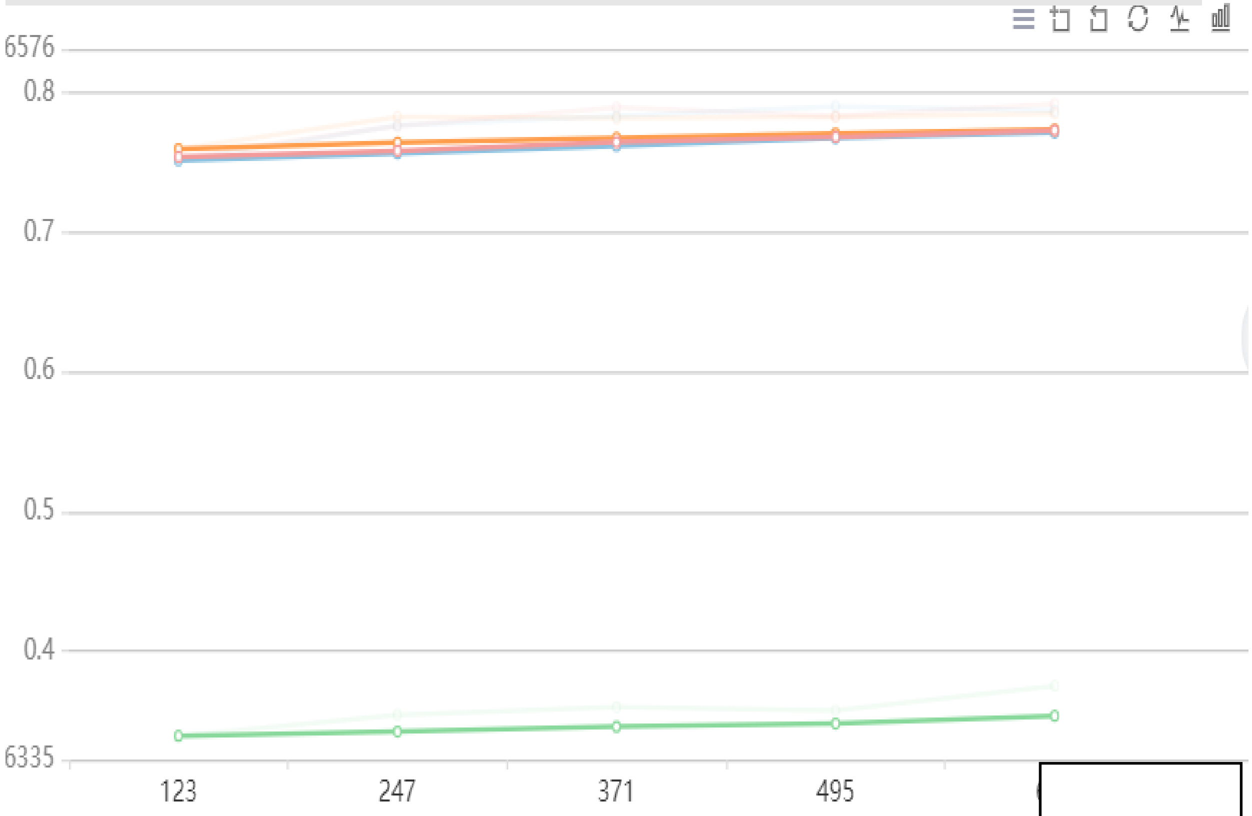
- Provided the most optimal hyperparameters, accuracy of model decreases by approximately 10% on with each imbalanced dataset.

Establishing a Baseline: How about Data Augmentation?

Training Loss



Validation Accuracy




```
train.py \
--datastore grid:cifar50-imbalance-01:2 \
--augment_data True \
--batch_size 64 \
--lr 2e-3
```

GRID.AI

Performs Very Poorly!

Model Summary

Dataset	Baseline	?	?	?	?	?
Full	0.792					
0.1	0.7296					
0.02	0.6304					
0.005	0.5446					
	Average Improvement:					

Experiment 1 Complete!

Model Summary

Dataset	Baseline	Artificial Balancing	?	?	?	?
Full	0.792					
0.1	0.7296					
0.02	0.6304					
0.005	0.5446					
Average Improvement:						

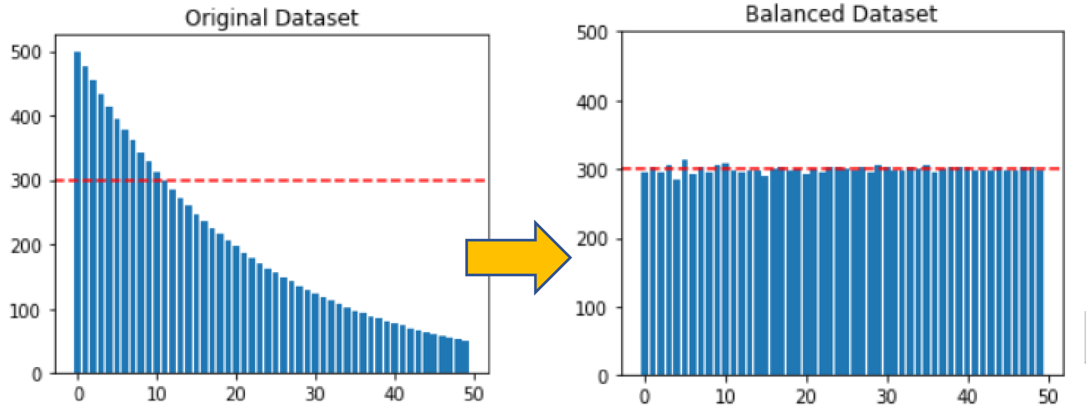
Experiment #2: Artificial Balancing

Motivation:

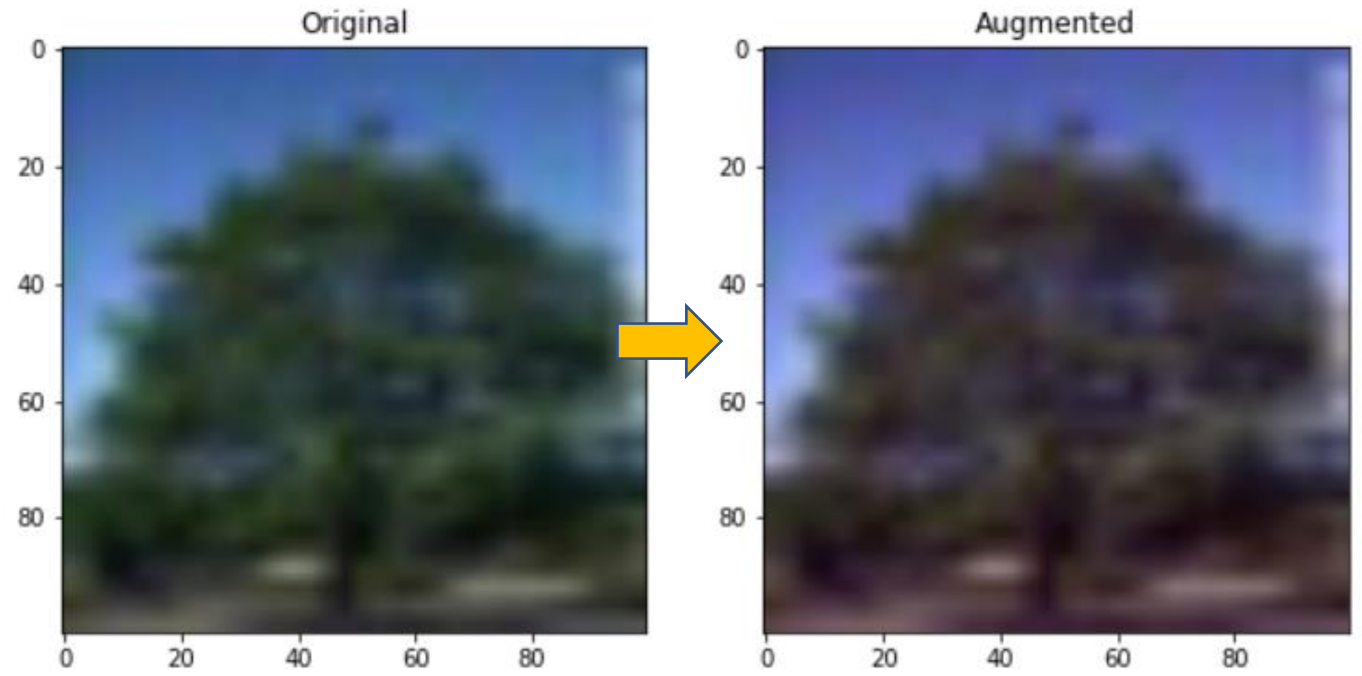
Shape the training dataset match the uniform test distribution.

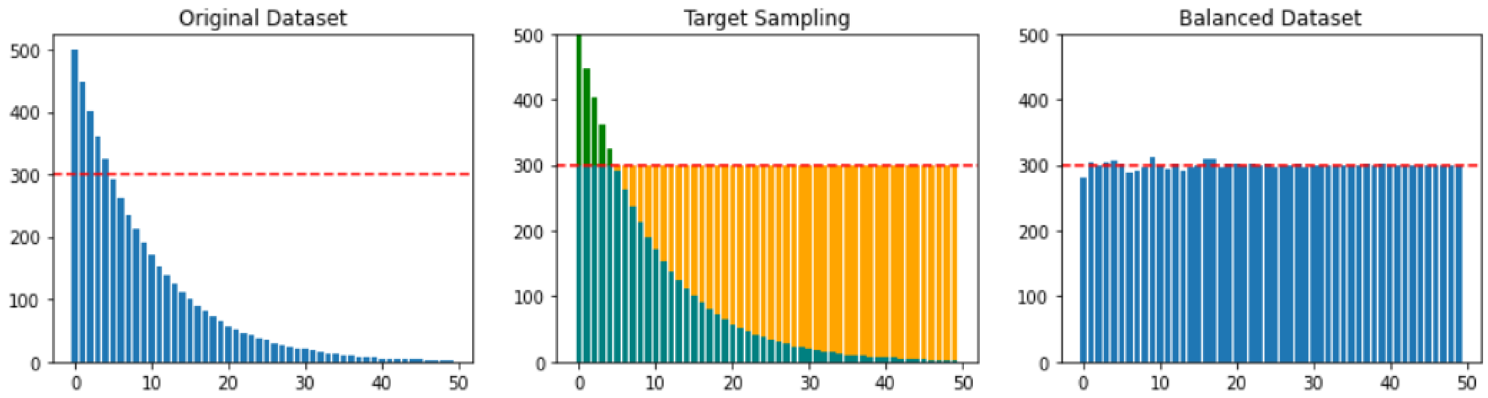
Procedures:

Downsampling and Upsampling with Data Augmentation.



Artificial Balancing: Implementation





Algorithm:

1) Calculate Histogram
2) Calculate Sampling Rate:
   Threshold Value / Count

Ex: Threshold Value = 300

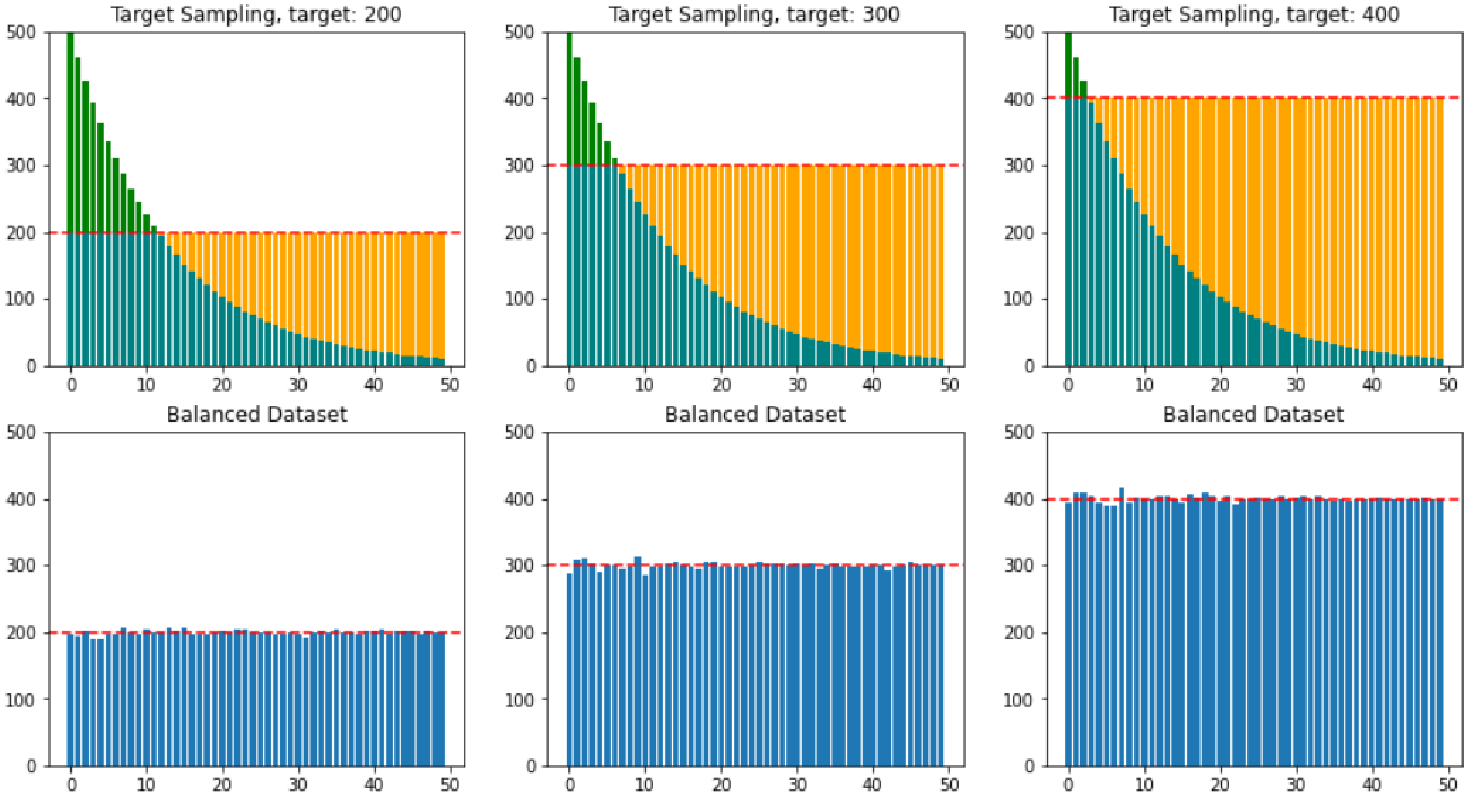
(Class = 0, Count = 500) => 300/500 => 0.6

(Class = 20, Count = 100) => 300/100 => 3

For decimal values, we generate a random number and perform down/up sampling if the number lies with the [0-0.x] range.

3) Iterate through dataset applying sampling rate.

Artificial Balancing: Implementation



Algorithm:

1) Calculate Histogram
2) Calculate Sampling Rate:
   Threshold Value / Count

Ex: Threshold Value = 300
(Class = 0, Count = 500) => 300/500 => 0.6
(Class = 20, Count = 100) => 300/100 => 3

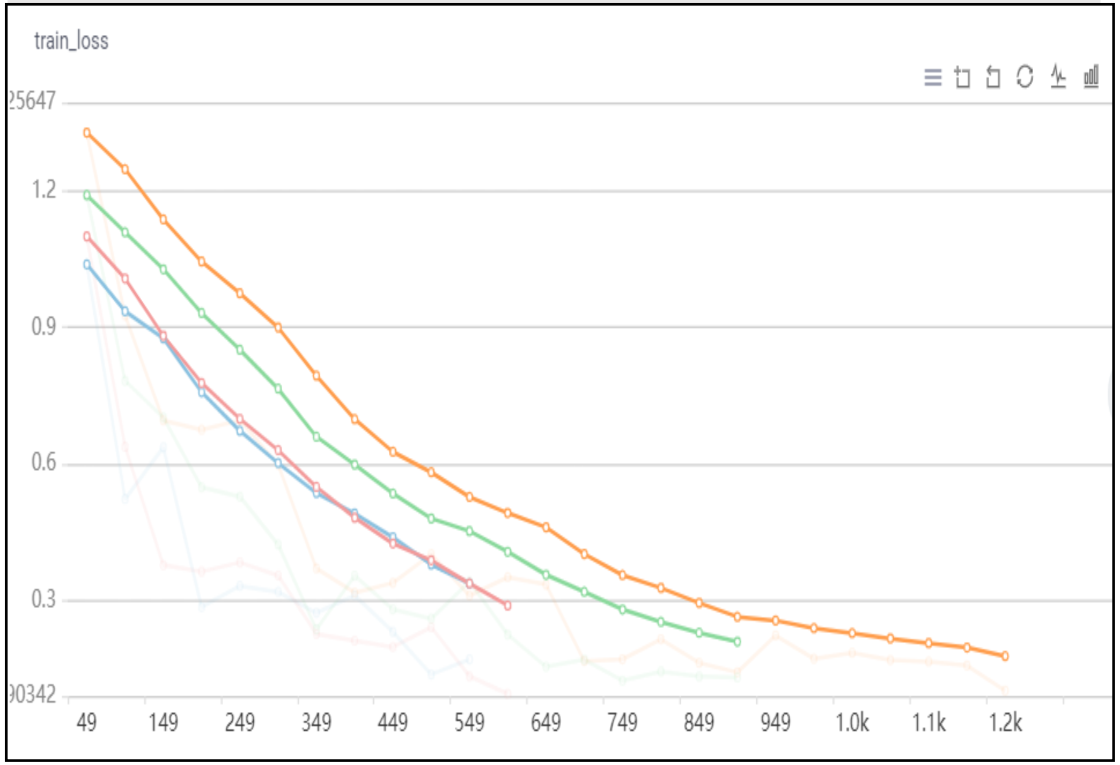
For decimal values, we generate a random number and perform down/up sampling if the number lies with the [0-0.x] range.

3) Iterate through dataset applying sampling rate.

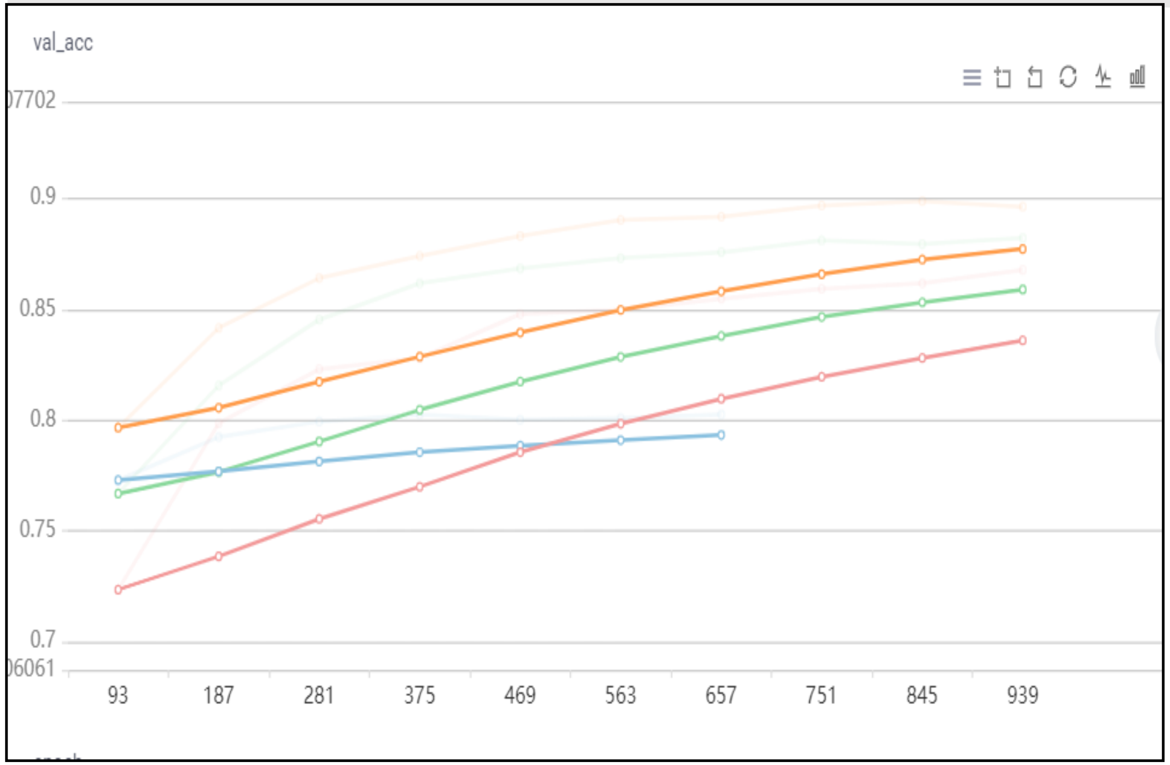
Threshold Value is also configurable and factored into hyperparameter sweep. In general, a low threshold produced higher performance, in which benefits of upsampling balanced benefits of downsampling.

Artificial Balancing: Training

Training Loss



Validation Accuracy




```
train.py \
--datastore grid:cifar50-balanced-0005:4 \
--batch_size 128 \
--lr 2e-3 \
--epochs 10
```

GRID.AI

Performs Better!

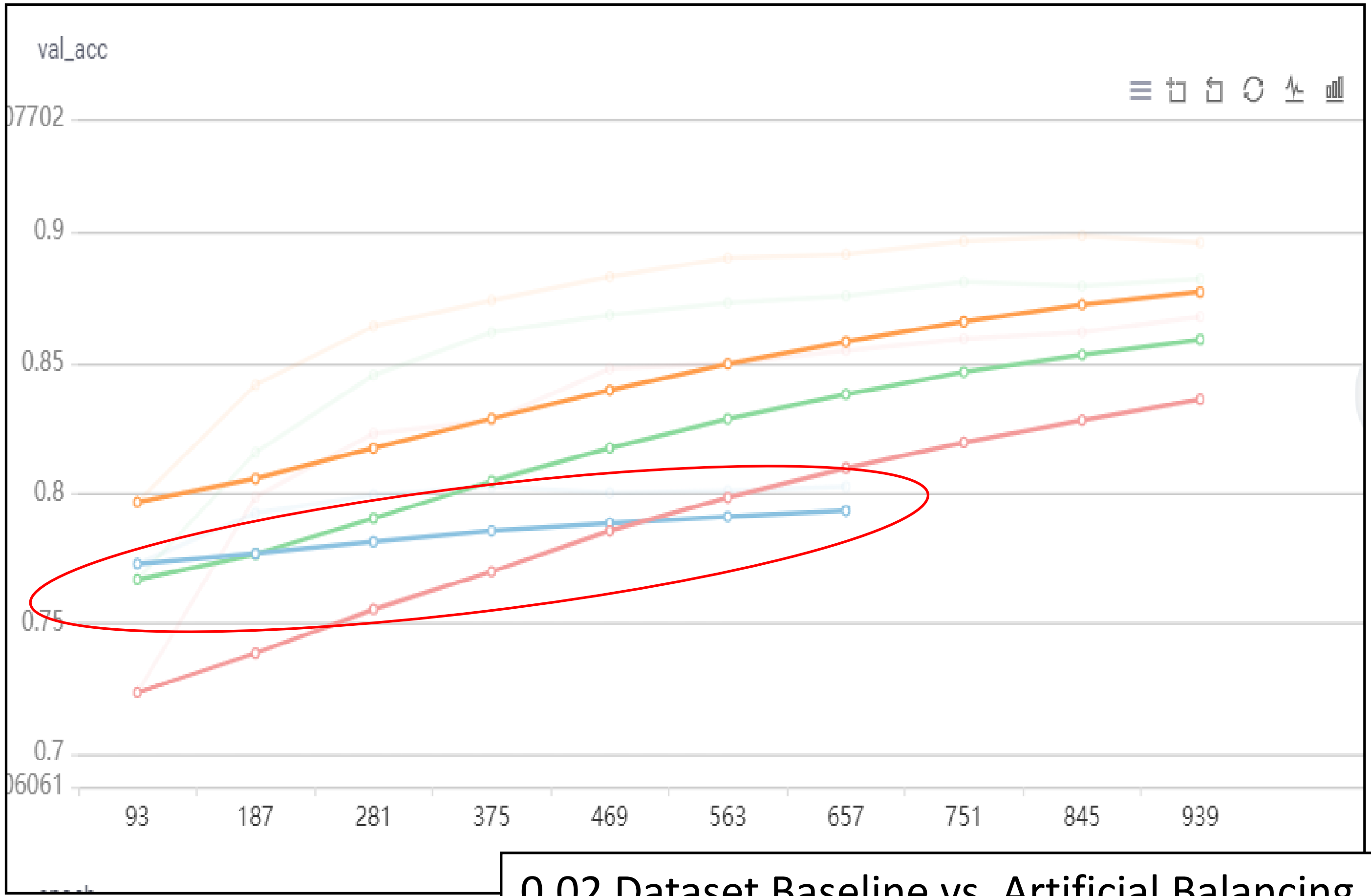
Artificial Balancing: Training

Results

Repeat Balancing for each dataset…

Dataset	Test Accuracy
Full	N/A
0.1	0.733
0.02	0.6788
0.005	0.5688

Average Improvement:	0.024



0.02 Dataset Baseline vs. Artificial Balancing at thresholds of [200, 300, 400].

All exceed previous accuracy!

Model Summary

Dataset	Baseline	Artificial Balancing	?	?	?	?
Full	0.792					
0.1	0.7296	0.733				
0.02	0.6304	0.6788				
0.005	0.5446	0.5688				
	Average Improvement:	0.024				

Experiment 2 Complete!

Artificial Balancing is labor-intensive and not a scalable solution. Can we do better?

Model Summary

Dataset	Baseline	Artificial Balancing	Weighted Loss	?	?	?
Full	0.792					
0.1	0.7296	0.733				
0.02	0.6304	0.6788				
0.005	0.5446	0.5688				
	Average Improvement:	0.024				

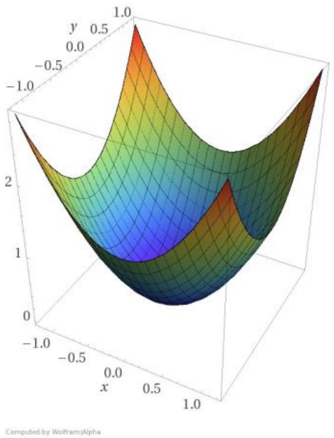
Experiment #3: Weighted Loss

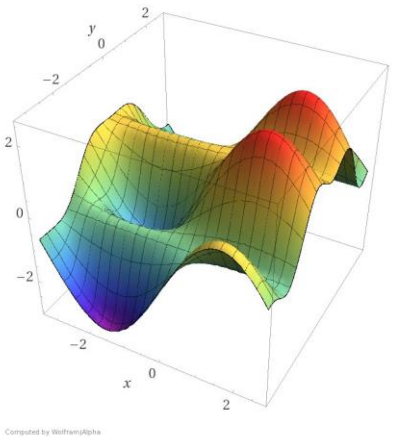
Motivation:

Emulate uniform training distribution without artificially balancing dataset.

Procedures:

Weighted Loss Equations





Weighted Loss: Implementation

Equations:

Inverse Number of Samples (INS)

$$w_{n,c} = \frac{1}{Number\ of\ Samples\ in\ Class\ c}$$

Inverse Square Number of Samples (ISNS)

$$w_{n,c} = \frac{1}{\sqrt[2]{Number\ of\ Samples\ in\ Class\ c}}$$

Effective Number of Samples (ENS)

$$w_{n,c} = \frac{1}{E_{n_c}}$$

$$E_{n_c} = \frac{1-\beta^{n_c}}{1-\beta}$$

“Class-Balanced Loss Based on Effective Number of Samples”, CVPR’19

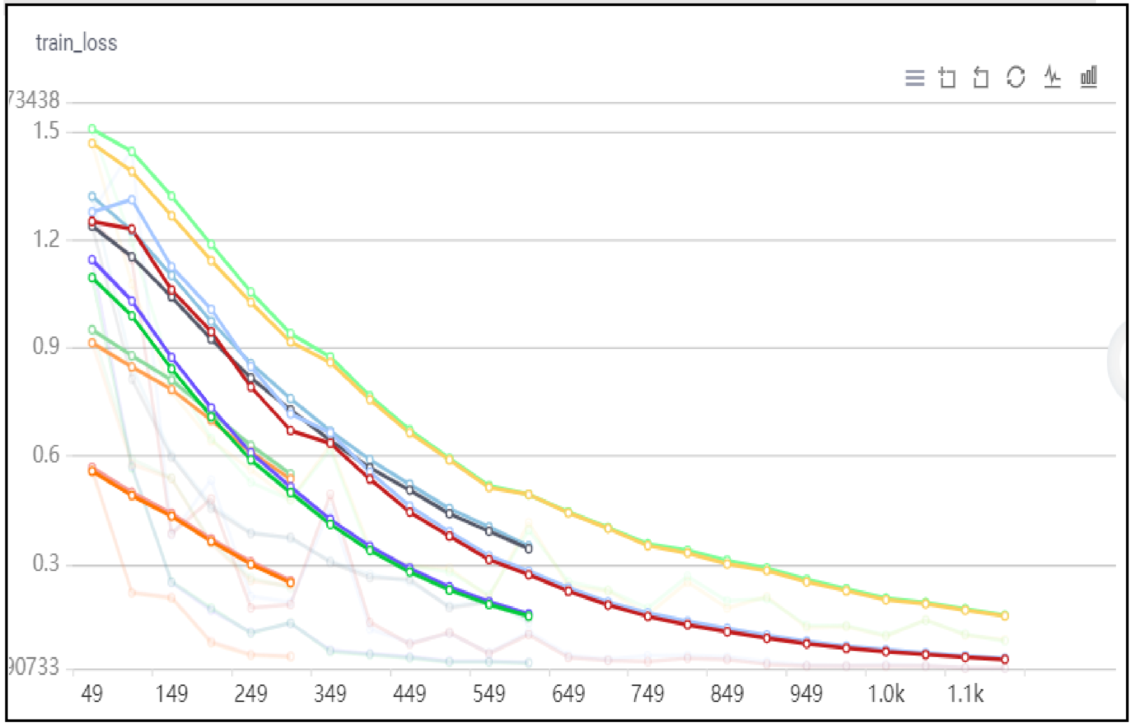
Code:

```
# Weighted Loss Functions
class WeightedLoss:
  def __init__(self, weighted_loss, hist, beta=None):
    # Store arguments, Calculate Normalized Weights
    if weighted_loss == "ins":
      weights = torch.tensor([1.0 / sample_count for sample_count in hist.values()])
      self.weight_map = weights / torch.sum(weights)

    elif weighted_loss == "isns":
      weights = torch.sqrt(torch.tensor([1.0 / sample_count for sample_count in hist.values()]))
      self.weight_map = weights / torch.sum(weights)

    elif weighted_loss == "ens":
      sample_counts = torch.tensor(list(hist.values()))
      e_numerator = 1.0 - torch.pow(beta, sample_counts)
      e_denominator = 1.0 - beta
      weights = e_denominator / e_numerator
      self.weight_map = weights / torch.sum(weights)

    else:  # Identity
      self.weight_map = torch.ones(50)

    # Store weights inside Cross Entropy Module
    self.loss = nn.CrossEntropyLoss(weight=self.weight_map.cuda())

  def __call__(self, logits, labels):
    return self.loss(logits, labels)
```

Simply Pass Weight Vectors to PyTorch Module

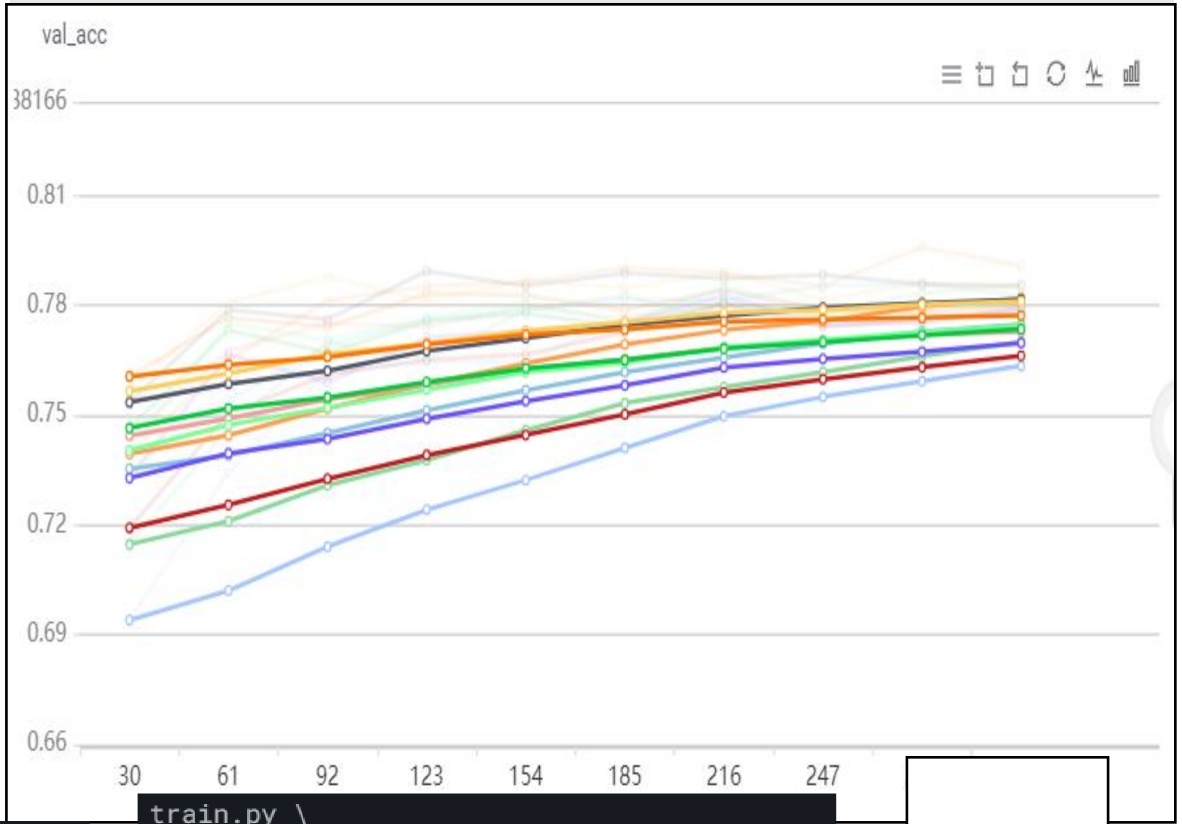
Weighted Loss: Loss Function Sweep

Training Loss




```
train.py \
--datastore grid:cifar50-imbalance-01:2 \
--batch_size "[64, 128, 256]" \
--lr "[8e-3, 2e-3]" \
--weighted_loss "['ins', 'isns']" \
--epochs 10
```

Validation Accuracy




```
train.py \
--datastore grid:cifar50-imbalance-01:2 \
--batch_size "[64, 128, 256]" \
--lr "[8e-3, 2e-3]" \
--epochs 10 \
--weighted_loss ens \
--beta "[0.99, 0.999]"
```

Repeated for each dataset.

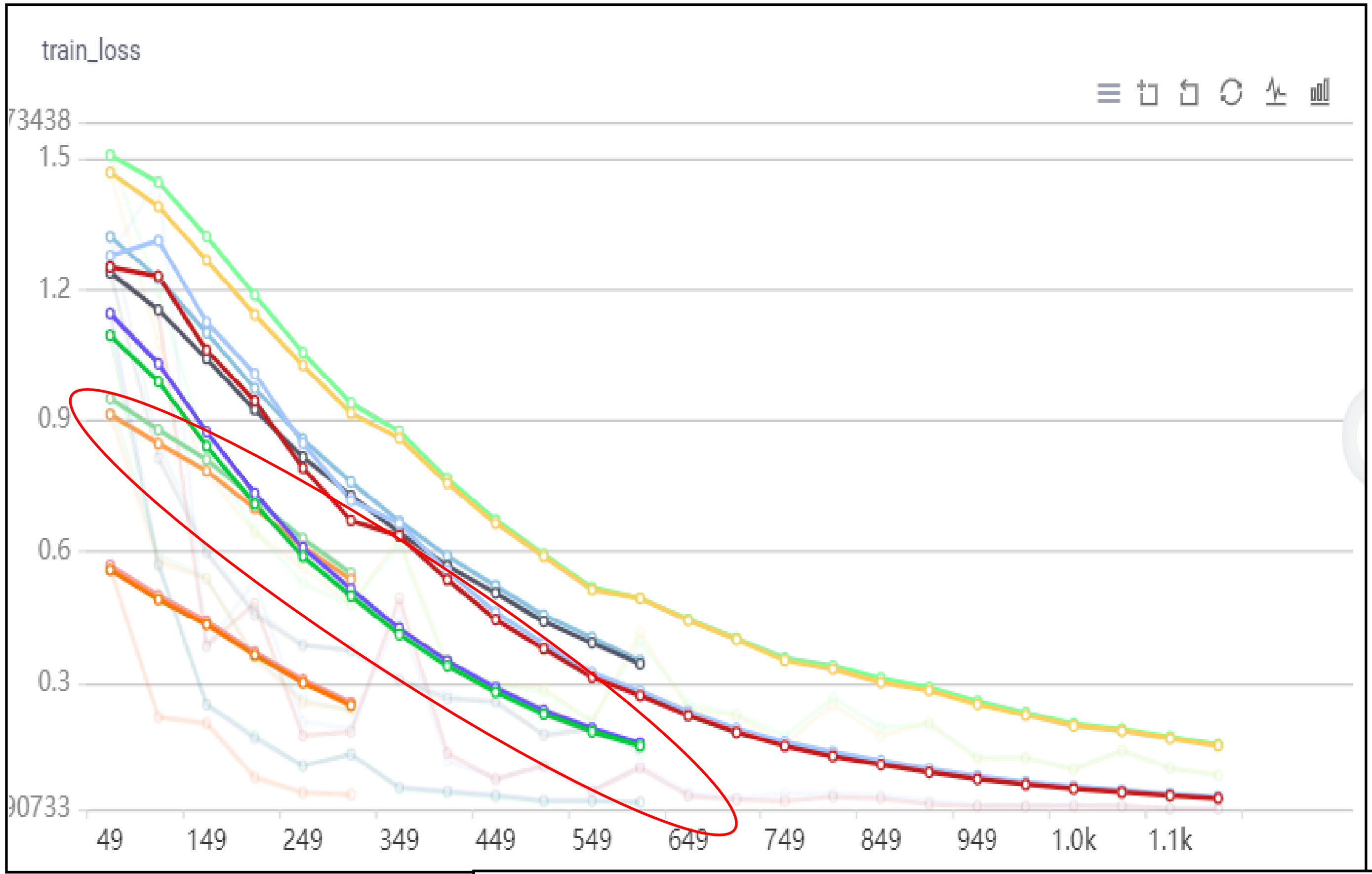
Weighted Loss: Loss Function Sweep

Results

Repeat Balancing for each dataset…

Dataset	Test Accuracy
Full	N/A
0.1	0.7458
0.02	0.6798
0.005	0.5352

Average Improvement:	0.019



0.01 Dataset trained on ins, isns, and ens loss functions. Comparable performance, no one weighted loss function is better in this case.

All exceed previous accuracy!

Model Summary

Dataset	Baseline	Artificial Balancing	Weighted Loss	?	?	?
Full	0.792					
0.1	0.7296	0.733	0.7458			
0.02	0.6304	0.6788	0.6798			
0.005	0.5446	0.5688	0.5352			
	Average Improvement:	0.024	0.019			

Experiment 3 Complete!

Weighted Loss serves as good substitute to Artificial Balancing.

Model Summary

Dataset	Baseline	Artificial Balancing	Weighted Loss	Bagging	?	?
Full	0.792					
0.1	0.7296	0.733	0.7458			
0.02	0.6304	0.6788	0.6798			
0.005	0.5446	0.5688	0.5352			
Average Improvement:		0.024	0.019			

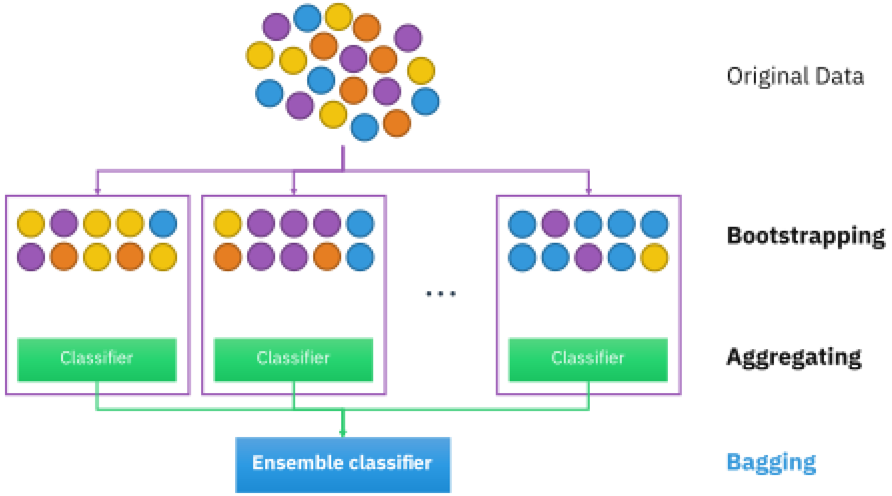
Experiment #4: Ensemble Training, Bagging

Motivation:

Combine multiple model copies to form more robust predictor.

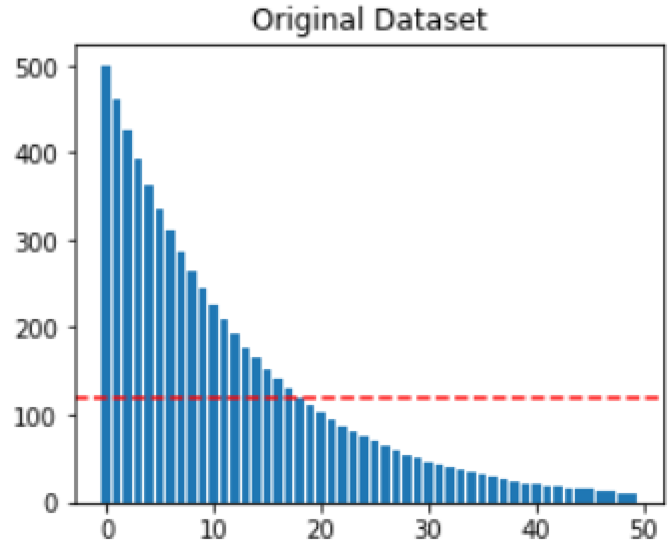
Procedures:

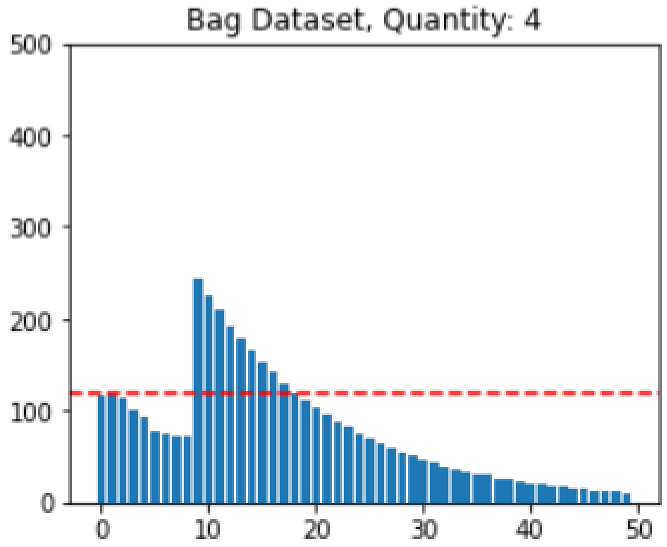
Balanced Dataset Creation

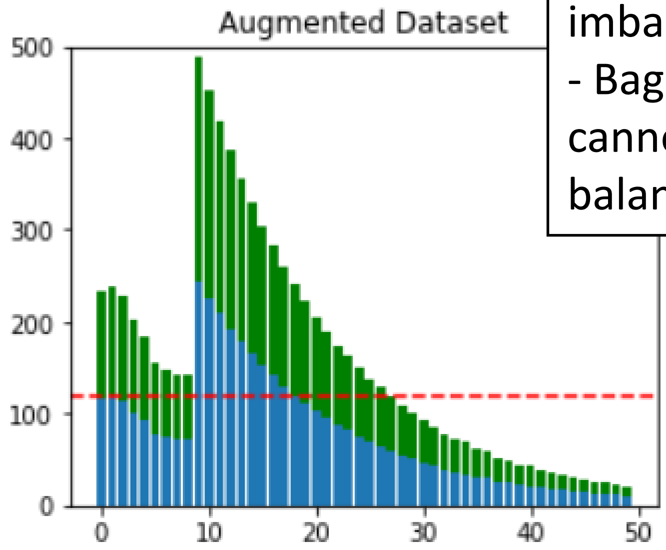


Bagging: Implementation

Dataset Curation

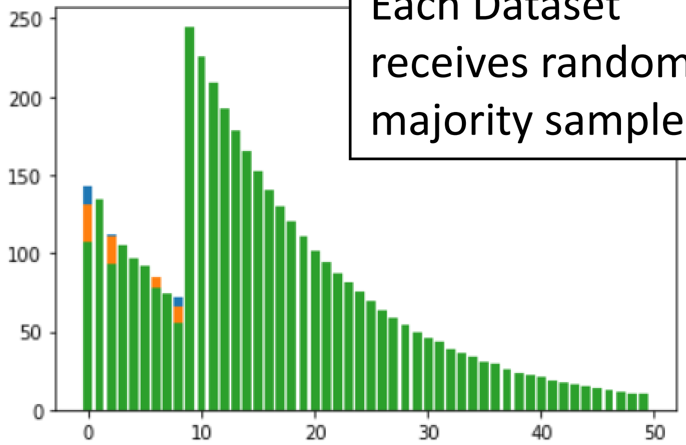






Comments:
- DA worsens class imbalance
- Bagged Datasets cannot be completely balanced.

Algorithm:
1) Compute Histogram on Training Dataset
2) Compute number of bags based on threshold value (max // threshold)
3) Split Dataset on Median
4) Copy samples in minority class to each bag
5) Deal out remaining samples from randomly shuffled majority class



Each Dataset receives random majority samples.

Bagging: Implementation

Code

```
def create_bags(args, imbalanced_json):
  # 1) Load Data
  entries = process_json(imbalanced_json)

  # 2) Calculate Number of Learners
  hist = get_histogram(entries)
  num_bags = max(hist.values()) // args['threshold']

  # 3) Create Datasets
  majority, minority = split_histogram(hist, entries, args['threshold'])
  random.shuffle(majority) # need shuffle the deck!
  bags = []
  for start in range(num_bags):
    current_bag = majority[start::num_bags] + minority
    bags.append(BaseDataset())

  return bags
```

1) Calculate Histogram

2) Compute Number of Bags

3) Split Histogram into Majority/Minority Classes

4) Shuffle and deal out decks

Bagging: Implementation

Code

```
class ModelEnsemble(LightningModule):
  def __init__(self, models):
    super().__init__()
    # Create Models
    self.models = models

    # For Metrics
    self.test_acc = torchmetrics.Accuracy()
    self.preds = []
    self.labels = []

  def forward(self, x):
    prediction = torch.zeros(50).cuda()
    weight = 1.0 / len(self.models)
    for model in self.models:
      model.eval()
      prediction = prediction + (weight * F.softmax(model(x), dim=1))
    return prediction
```

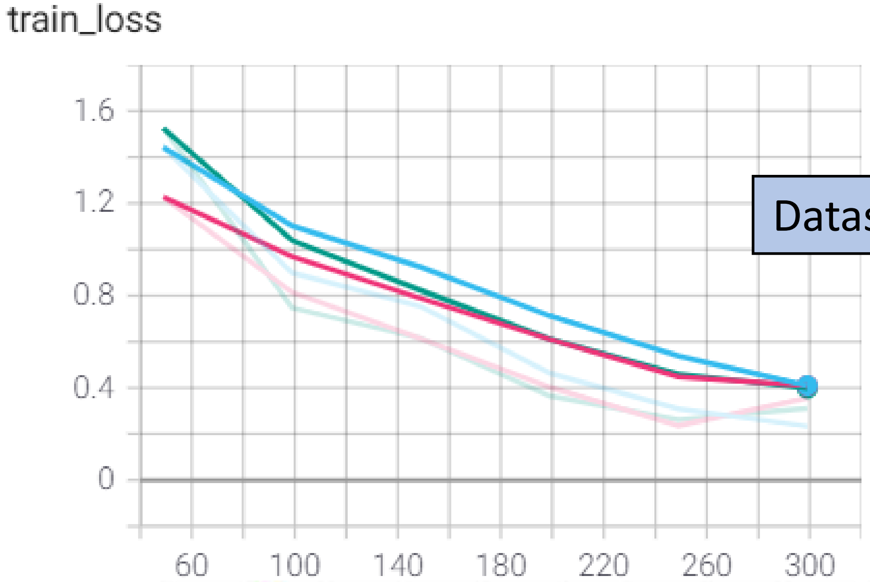
1) Accept List of Trained Models

2) Forward Pass performs simple average over softmax output

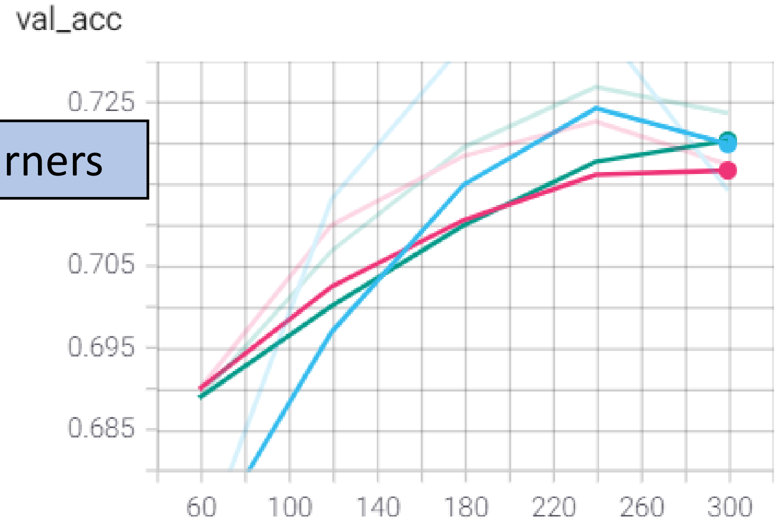
Bagging: Training

Training Loss

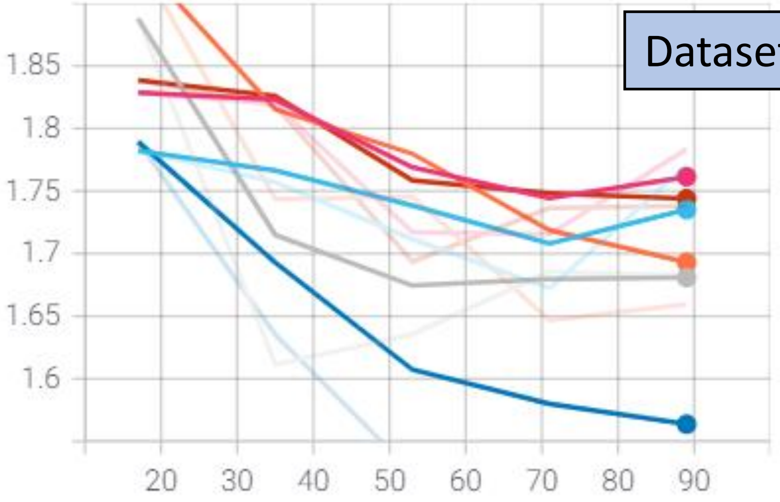
Validation Accuracy

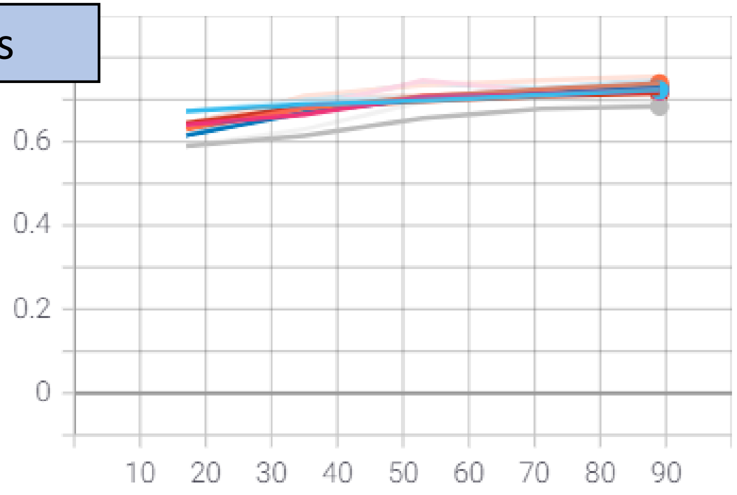


Dataset: 0.1 - 3 Learners



Dataset: 0.005 - 5 Learners





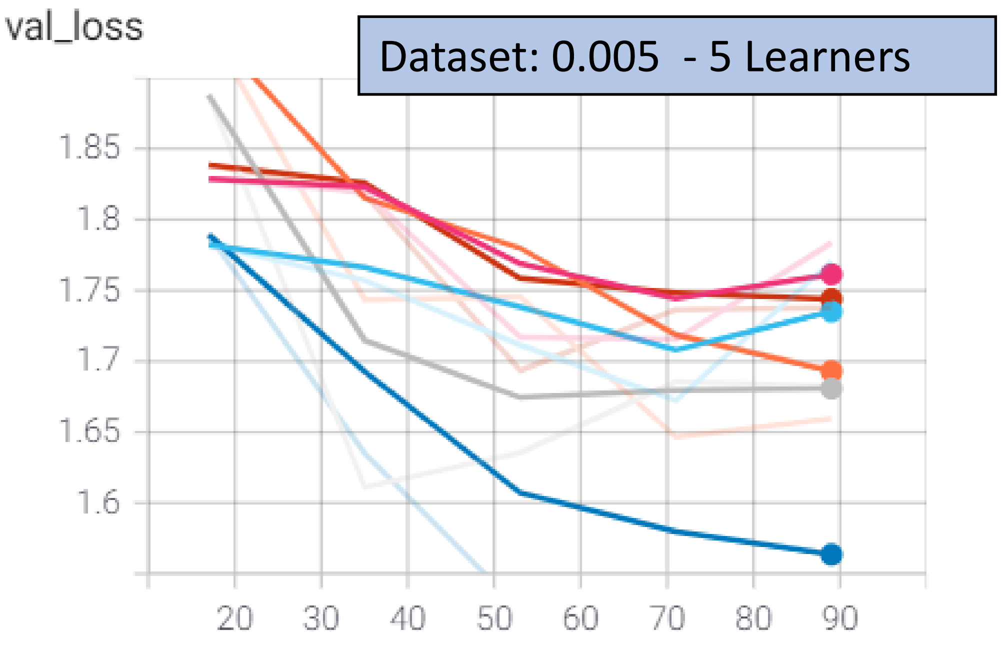
Bagging: Training

Results

Repeat Balancing for each dataset…

Dataset	Test Accuracy	Average Learner Accuracy
Full	N/A	N/A
0.1	0.7350	0.72
0.02	0.6760	0.65
0.005	0.5454	0.53

Average Improvement:	0.019



Each learner descends down objective function in different directions and ends up at different loss. Combined learners make for stronger predictor.

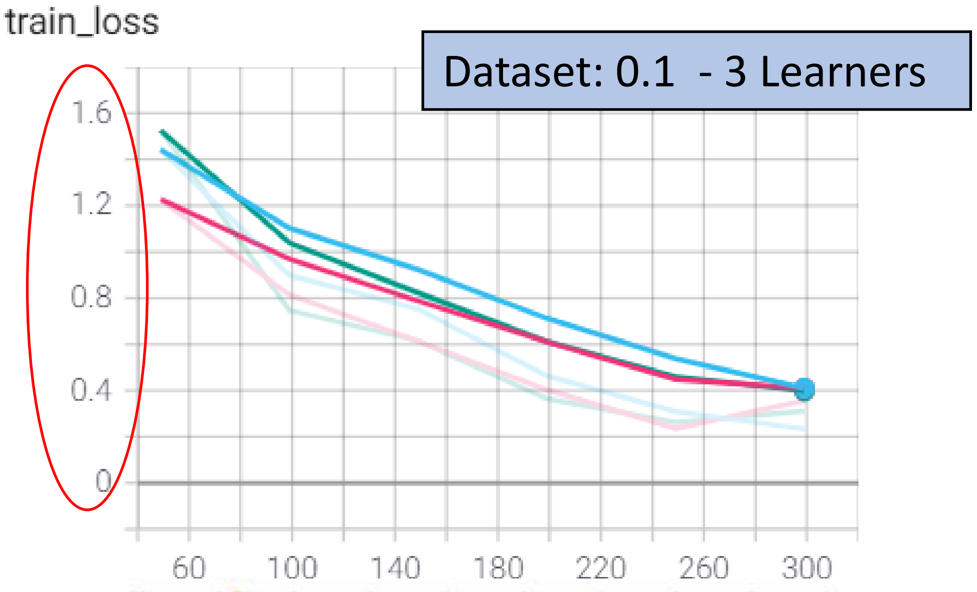
Bagging: Training

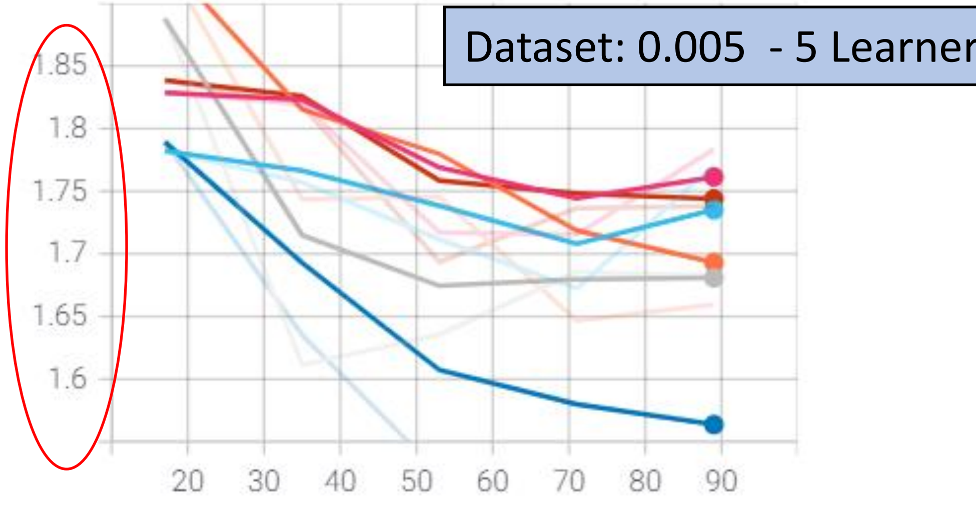
Results

Repeat Balancing for each dataset…

Dataset	Test Accuracy	Average Learner Accuracy
Full	N/A	N/A
0.1	0.7350	0.72
0.02	0.6760	0.65
0.005	0.5454	0.53

Average Improvement:	0.019





Potential to learn more with more data.

Model Summary

Dataset	Baseline	Artificial Balancing	Weighted Loss	Bagging / WL	?	?
Full	0.792					
0.1	0.7296	0.733	0.7458	0.7350		
0.02	0.6304	0.6788	0.6798	0.6760		
0.005	0.5446	0.5688	0.5352	0.5454		
	Average Improvement:	0.024	0.019	0.015		

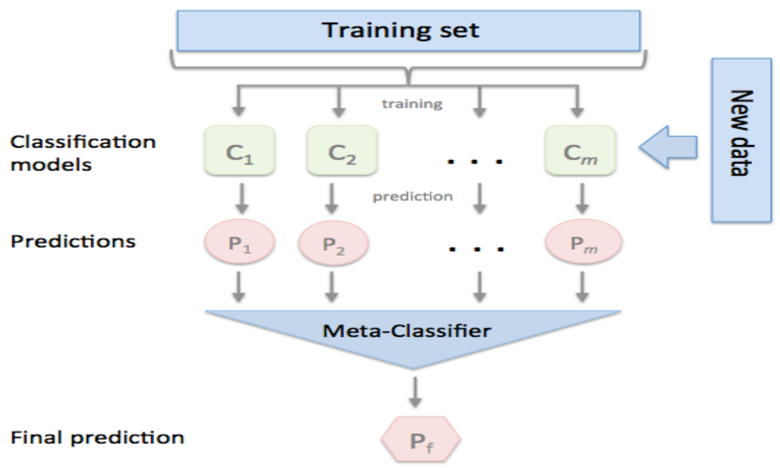
Experiment 4 Complete!

Ensemble Techniques show promise with more data. Let's give each model more data.

Model Summary

Dataset	Baseline	Artificial Balancing	Weighted Loss	Bagging / WL	Stacking	?
Full	0.792					
0.1	0.7296	0.733	0.7458	0.7350		
0.02	0.6304	0.6788	0.6798	0.6760		
0.005	0.5446	0.5688	0.5352	0.5454		
	Average Improvement:	0.024	0.019	0.015		

Experiment #5: Ensemble Training, Stacking



Motivation:

Apply ensemble techniques to entire dataset.
Additionally, create more diverse ensemble of learners.

Procedures:

Model Architecture Ensemble.

Stacking: Implementation

Code

```
hps_full = {'batch_size': 128, "lr": 2e-3, "epochs": 5,
            "weighted_loss": None, "output_dir": 'stacking-01'}

# 0) For reference
seed.seed_everything(1)

# 1) Init Ensemble Components
model_types = ['efficientnet_b0', 'efficientnet_b1', 'efficientnet_b2']
trainer = Trainer(gpus=1, max_epochs=hps_full['epochs'], logger=TensorBoardLogger(save_dir=hps_full['output_dir']))
dataset = BaseDataset(train_json)

# 2) Train Models
models = []
for i in range(3):
  train_dataloader, val_dataloader = get_dataloaders(dataset, batch_size=hps_full['batch_size'])
  test_dataloader = get_test_dataloader(test_json)
  hist = get_histogram(process_json(train_json))
  index = i % len(model_types)
  model = BaseNet(model_type=model_types[index], lr=hps_full['lr'], weighted_loss=hps_full['weighted_loss'], hist=hist)
  trainer = Trainer(gpus=1, max_epochs=hps_full['epochs'], logger=TensorBoardLogger(save_dir=hps_full['output_dir']))
  trainer.fit(model, train_dataloader, val_dataloader)
  models.append(model)
  trainer.test(model, test_dataloader)

# 3) Combine Models and Inference
test_dataloader = get_test_dataloader(test_json)
for model in models:
  if torch.cuda.is_available():
    model.cuda()
ensemble = ModelEnsemble(models)
trainer.test(ensemble, test_dataloader)
```

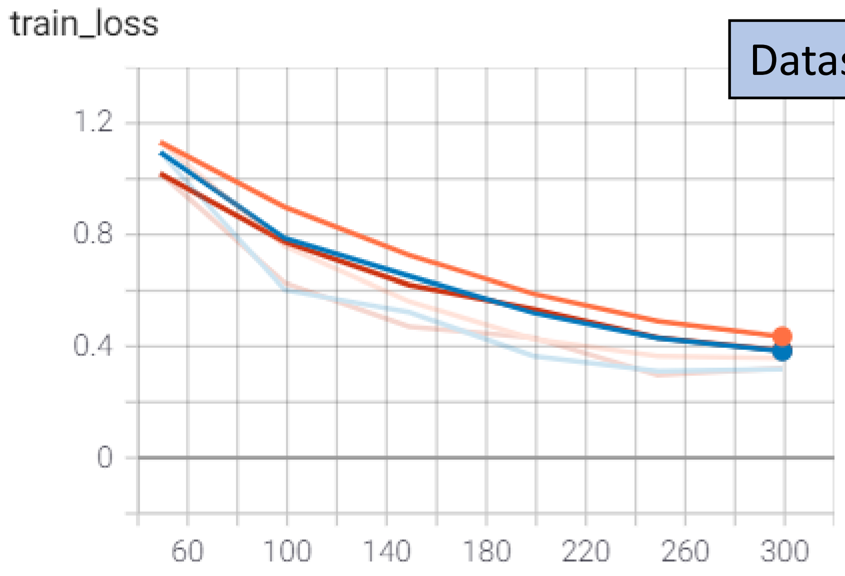
Same training procedure as bagging, except for two changes:

1) Each model trains on entire dataset
2) Each model is initialized under different model architecture.

Three Model Architectures selected were EfficientNets 0-2.

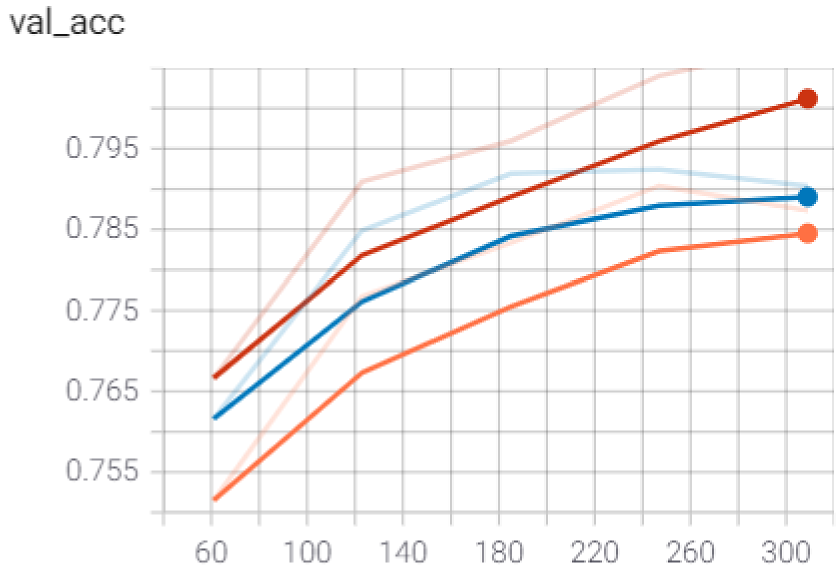
Stacking: Training

Training Loss



Dataset: 0.1

Validation Accuracy



Unlike Bagging, which had a variable number of learners for each balanced sub-dataset, all datasets using stacking had 3 learners corresponding to each model architecture.

Stacking: Training

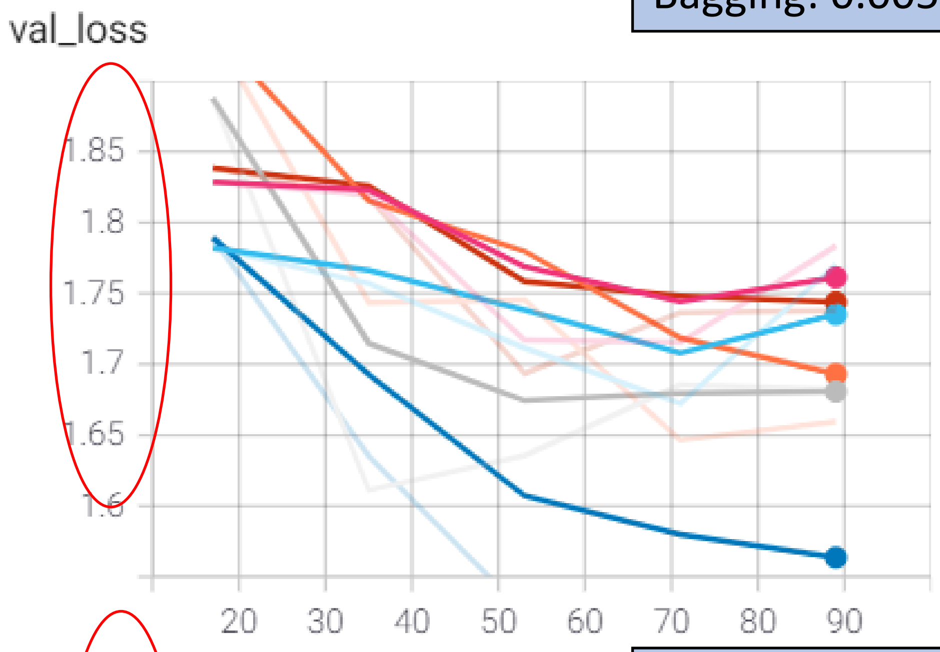
Results

Repeat Balancing for each dataset…

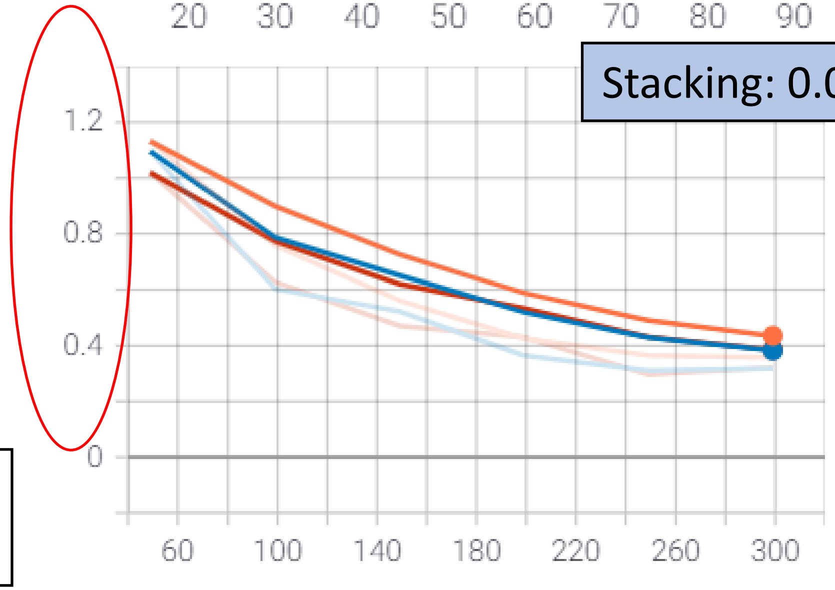
Dataset	Test Accuracy	Average Learner Accuracy
Full	N/A	N/A
0.1	0.7720	0.7335
0.02	0.6614	0.6360
0.005	0.5626	0.5436

Average Improvement:	0.030

Bagging: 0.005



Stacking: 0.005



More data produces stronger learner base.

Model Summary

Dataset	Baseline	Artificial Balancing	Weighted Loss	Bagging / WL	Stacking	?
Full	0.792					
0.1	0.7296	0.733	0.7458	0.7350	0.7720	
0.02	0.6304	0.6788	0.6798	0.6760	0.6614	
0.005	0.5446	0.5688	0.5352	0.5454	0.5626	
	Average Improvement:	0.024	0.019	0.015	0.030	

Experiment 5 Complete!

All that is left is to turn on individual model balancing.

Model Summary

Dataset	Baseline	Artificial Balancing	Weighted Loss	Bagging / WL	Stacking	?
Full	0.792					
0.1	0.7296	0.733	0.7458	0.7350	0.7720	
0.02	0.6304	0.6788	0.6798	0.6760	0.6614	
0.005	0.5446	0.5688	0.5352	0.5454	0.5626	
	Average Improvement:	0.024	0.019	0.015	0.030	

Experiment #6: Ensemble Training, Stacking with Weighted Loss

Motivation:

Combine successes of weighted loss and stacking ensemble technique.

Procedures:

Simply repeat last experiment turning on weighted loss!

Stacking with Weighted Loss: Training

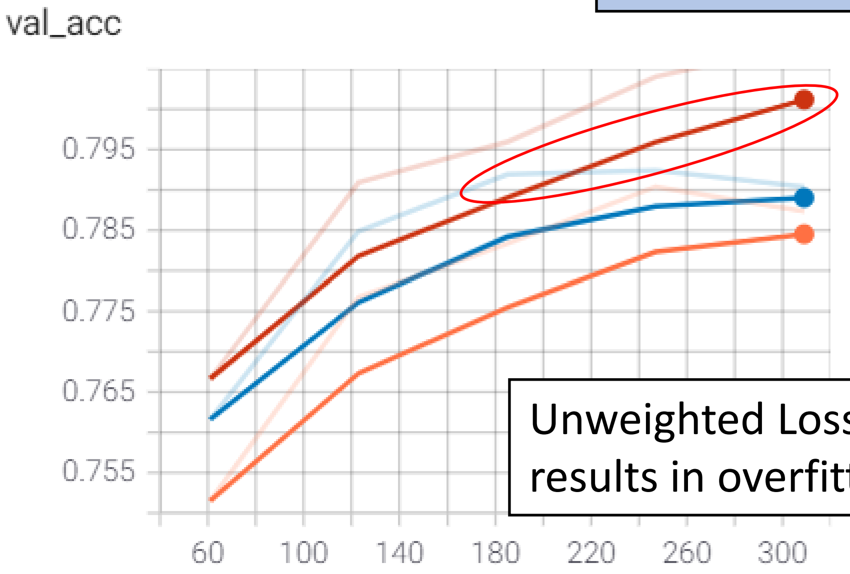
Results

Repeat Balancing for each dataset…

Dataset	Test Accuracy	Average Learner Accuracy
Full	N/A	N/A
0.1	0.7933	0.7482
0.02	0.7253	0.6935
0.005	0.6218	0.5997

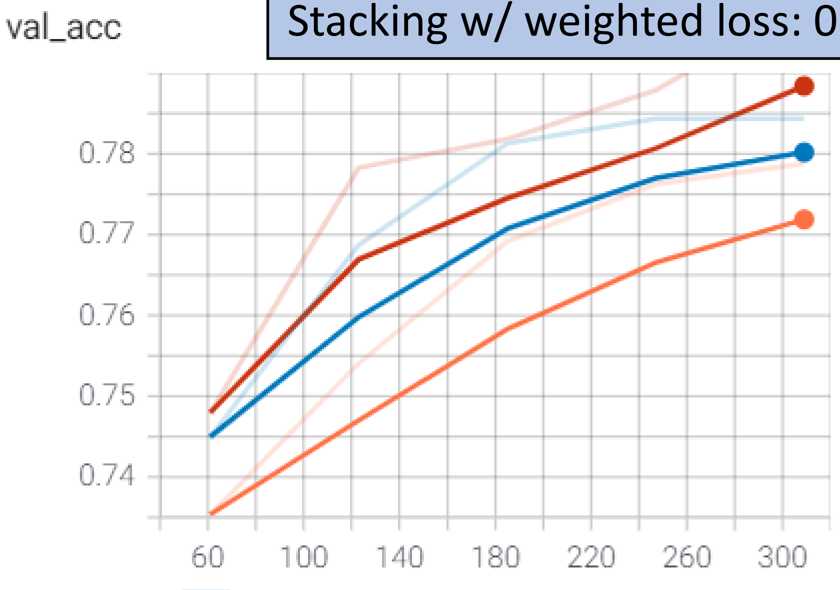
Average Improvement:	0.079

Stacking: 0.1



Unweighted Loss results in overfitting.

Stacking w/ weighted loss: 0.1



Model Summary

Dataset	Baseline	Artificial Balancing	Weighted Loss	Bagging / WL	Stacking	Stacking / WL
Full	0.792					
0.1	0.7296	0.733	0.7458	0.7350	0.7720	0.7933
0.02	0.6304	0.6788	0.6798	0.6760	0.6614	0.7253
0.005	0.5446	0.5688	0.5352	0.5454	0.5626	0.6218
Average Improvement:		0.024	0.019	0.015	0.030	0.079

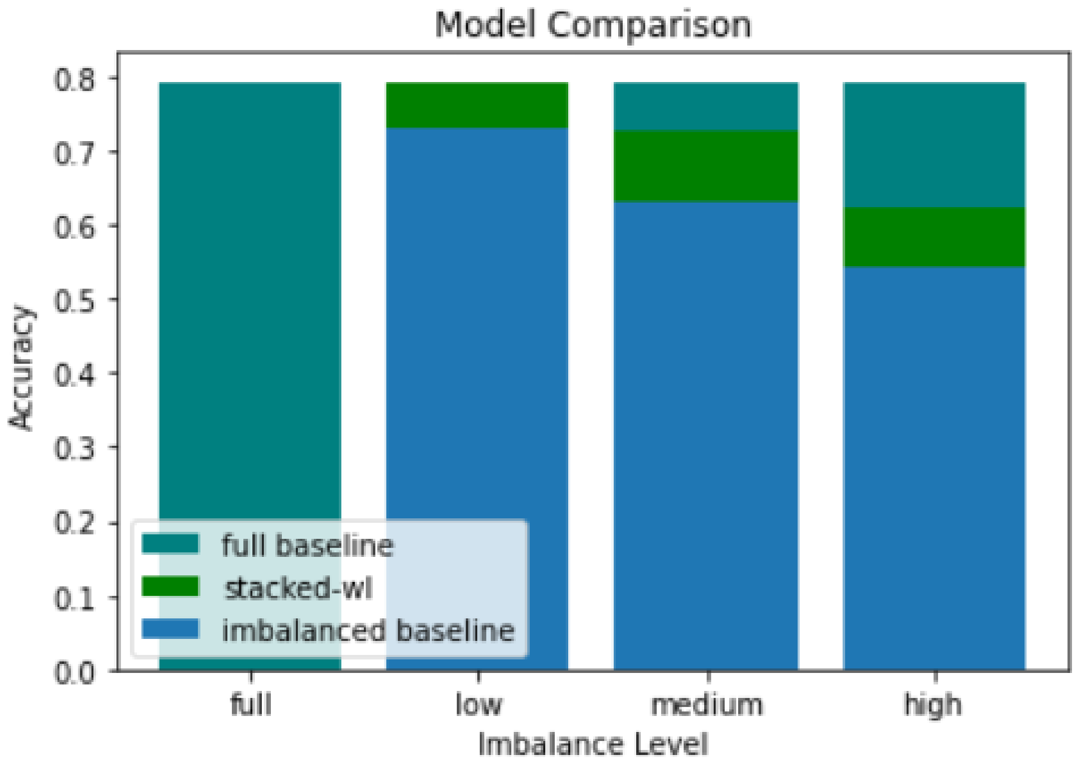
Experiment 6 Complete!

Model Summary

Dataset	Baseline	Artificial Balancing	Weighted Loss	Bagging / WL	Stacking	Stacking / WL
Full	0.792					
0.1	0.7296	0.733	0.7458	0.7350	0.7720	0.7933
0.02	0.6304	0.6788	0.6798	0.6760	0.6614	0.7253
0.005	0.5446	0.5688	0.5352	0.5454	0.5626	0.6218
	Average Improvement:	0.024	0.019	0.015	0.030	0.079

Best Results

Dataset	Baseline Accuracy	Maximum Accuracy	Improvement	Modeling Technique
0.1	0.7296	0.793	0.064	Stacking/WL
0.02	0.6304	0.725	0.095	Stacking/WL
0.005	0.5446	0.622	0.077	Stacking/WL



Model Name	Test Accuracy	# Parameters
efficientnet_b2	0.804	10 M
resnet50d	0.793	22 M
efficientnet_b1	0.7892	8 M
efficientnet_b0	0.788	5 M
densenet121	0.770	20 M
resnet50	0.758	22 M
resnet34d	0.730	20 M

Model Ensemble = 23 M
ResNet50 = 22 M !

Best Results

Dataset	Baseline Accuracy	Maximum Accuracy	Improvement	Modeling Technique
0.1	0.7296	0.793	0.064	Stacking/WL
0.02	0.6304	0.725	0.095	Stacking/WL
0.005	0.5446	0.622	0.077	Stacking/WL

Model Summary

Dataset	Baseline	Artificial Balancing	Weighted Loss	Bagging / WL	Stacking	Stacking / WL
Full	0.792					
0.1	0.7296	0.733	0.7458	0.7350	0.7720	0.7933
0.02	0.6304	0.6788	0.6798	0.6760	0.6614	0.7253
0.005	0.5446	0.5688	0.5352	0.5454	0.5626	0.6218
	Average Improvement:	0.024	0.019	0.015	0.030	0.079

Thank You!