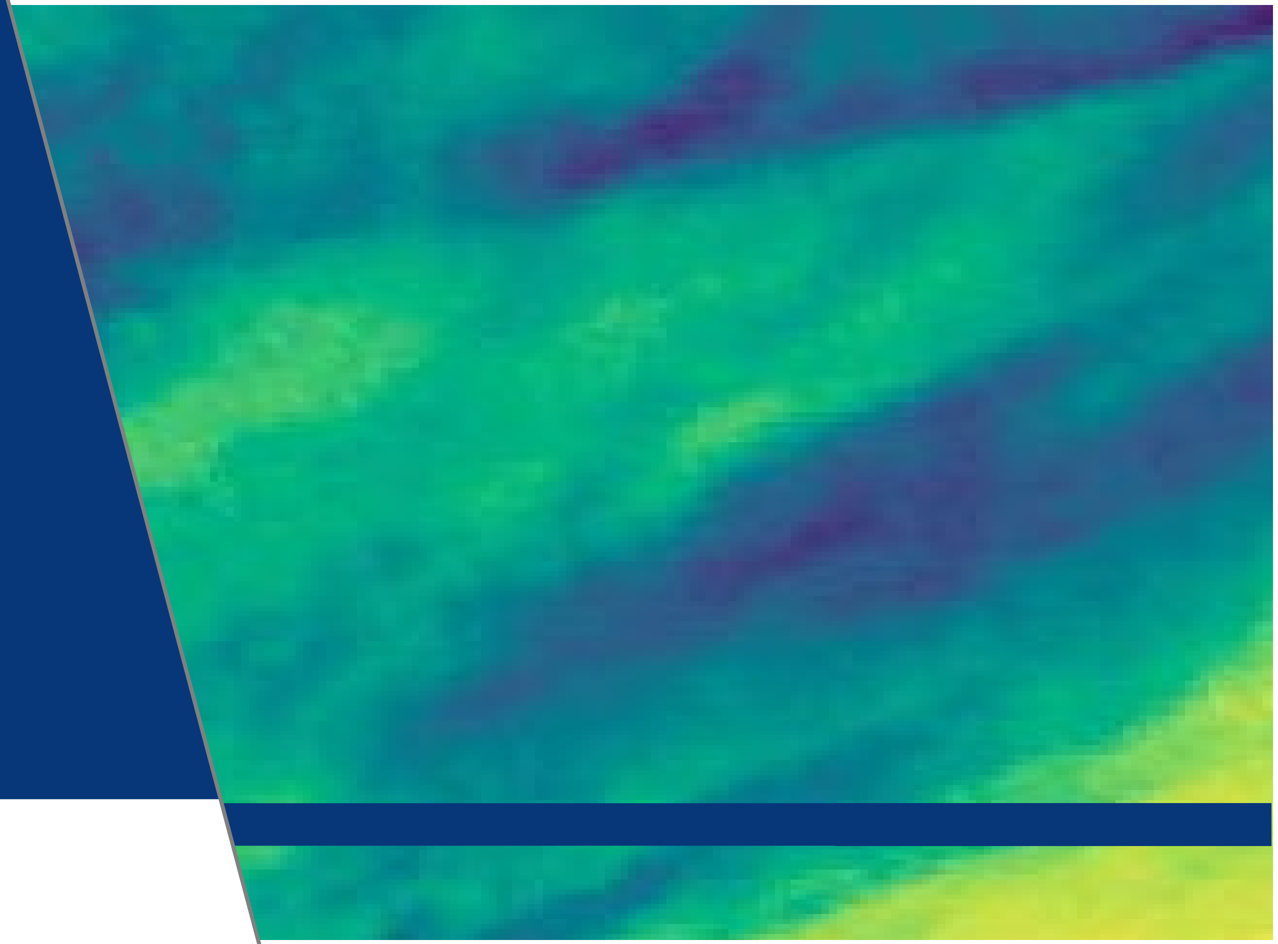
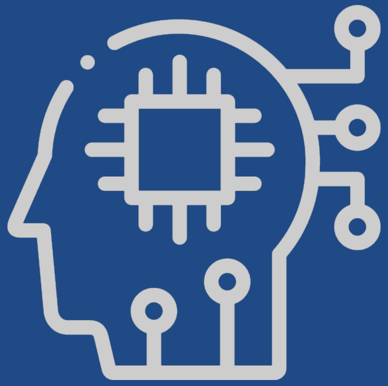
COMP4211 Final Project

Project 1

Jasper | Ryder | Yerke | Daniel





The Problem

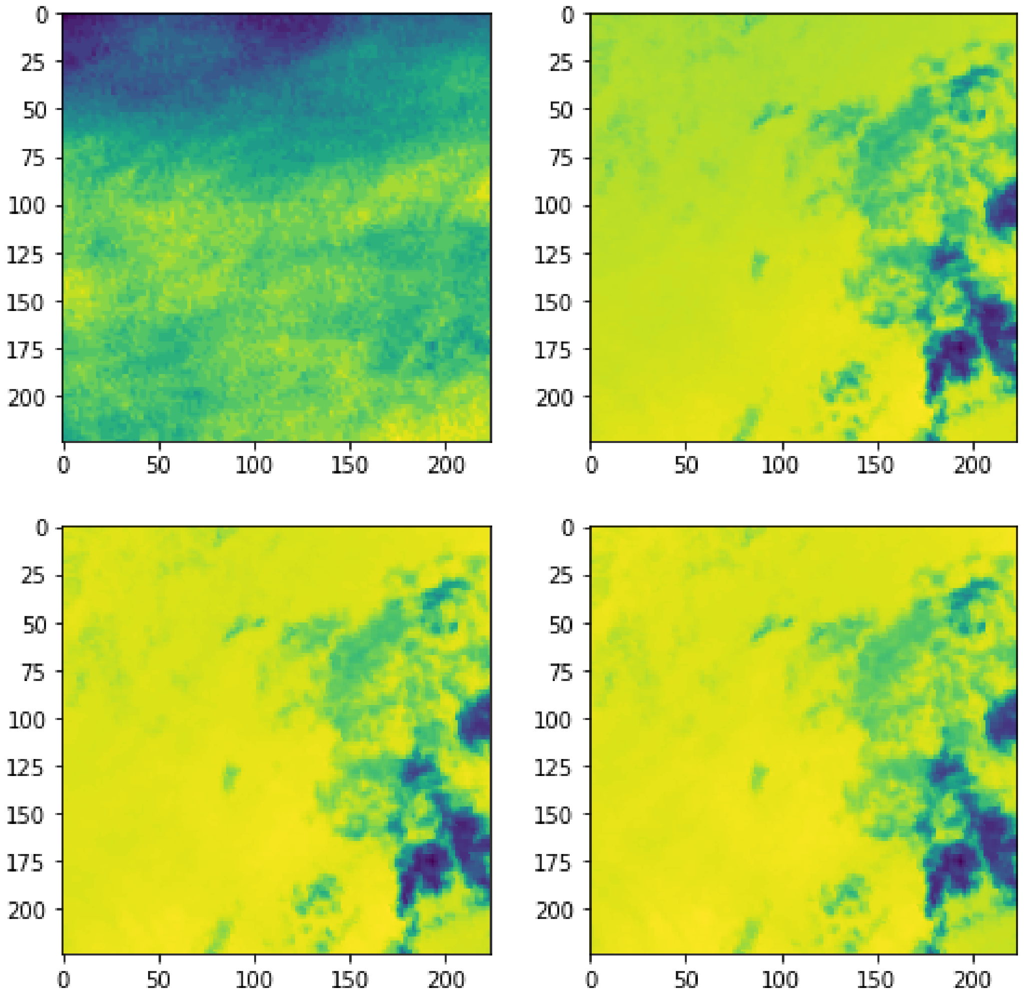
What needs to be done?

Design Goals

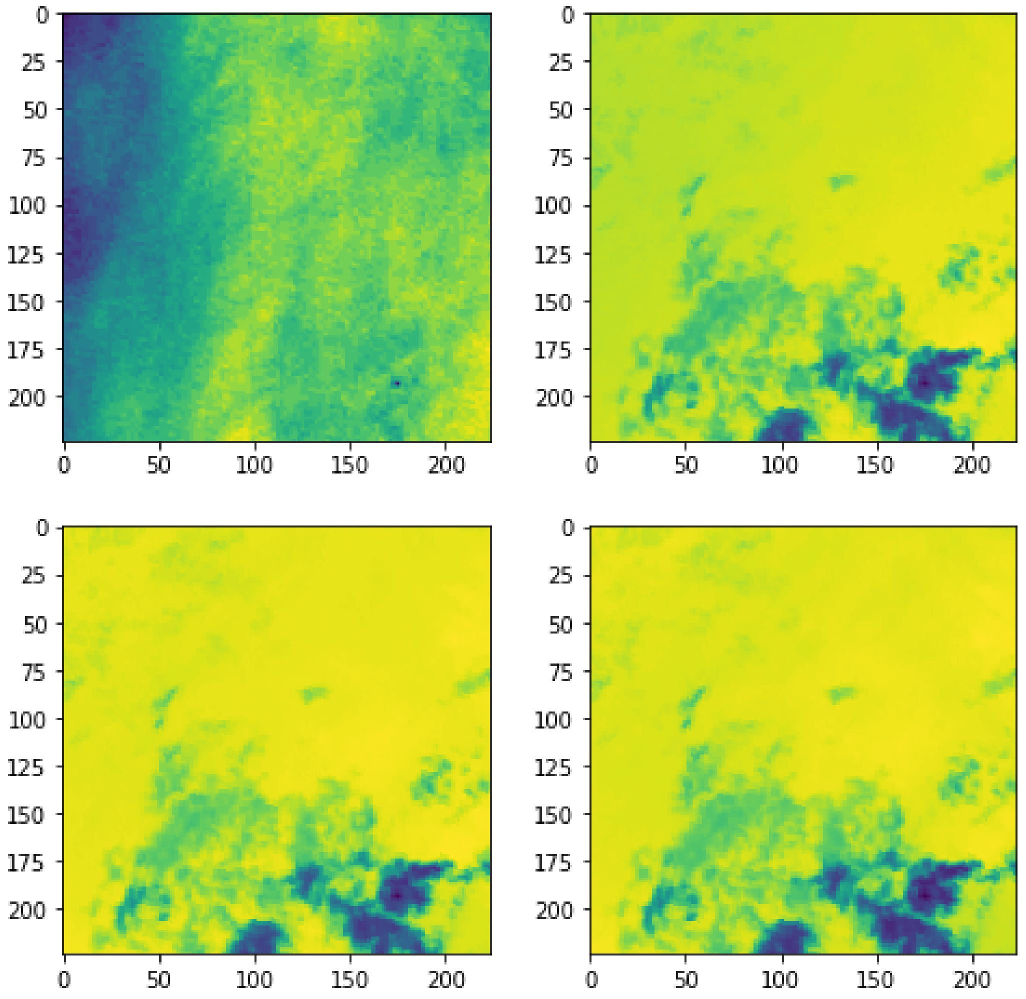
1. Robust
2. Good Precision
3. Good Recall
4. Good F1 Score

Data Augmentation

Original Data



Augmented Data



The First Task - Overview

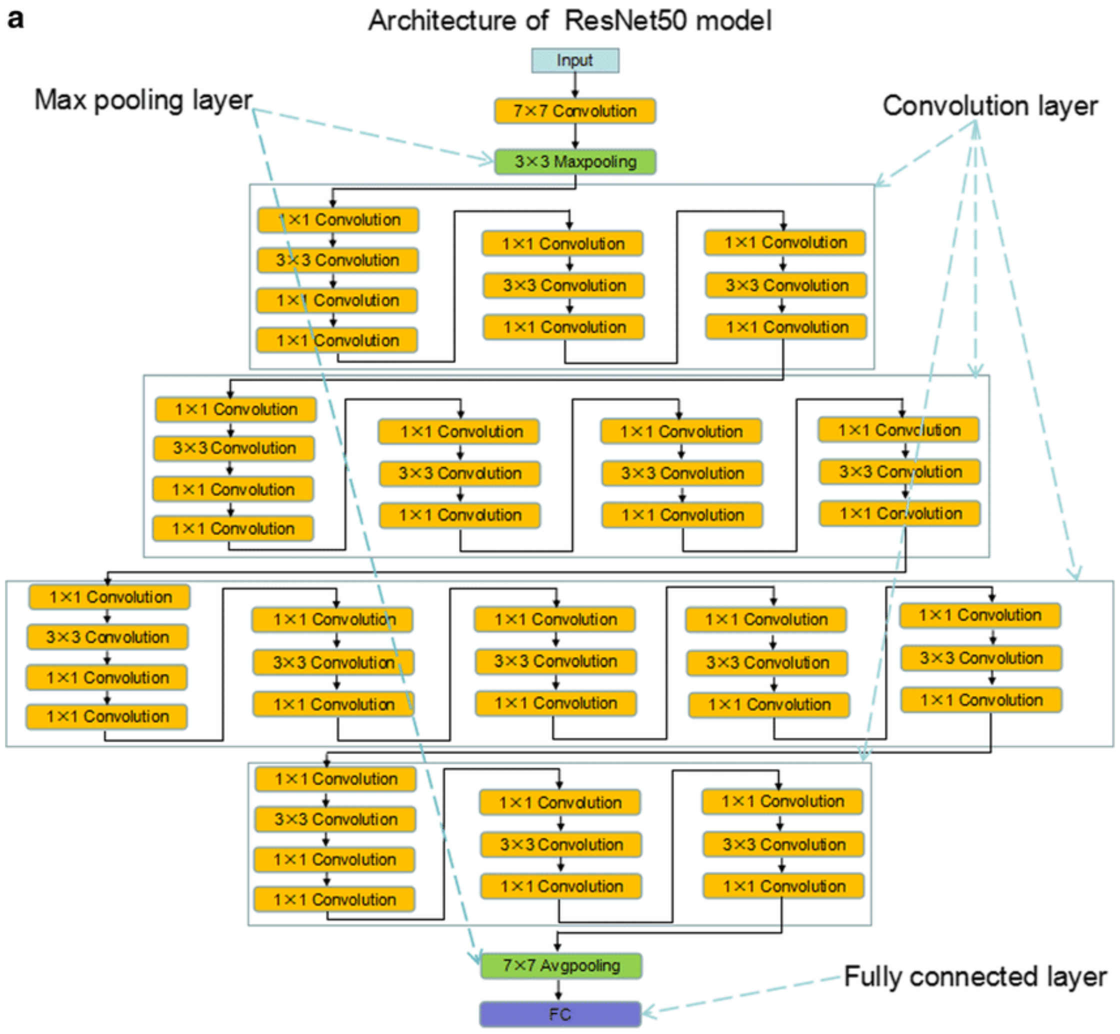
- To build a CNN model using cross-entropy loss

Model Specifications

- ResNet50 Model
- Stochastic Gradient Descent:
  - Learning Rate = 0.001
  - Momentum = 0.9
- Uses cross-entropy loss function

$$L(x)_{CE} = -\sum_{j=1}^{n} t_j \log(p_j)$$
$$= -t_i \log(p_i)$$
$$= -\log(p_i)$$

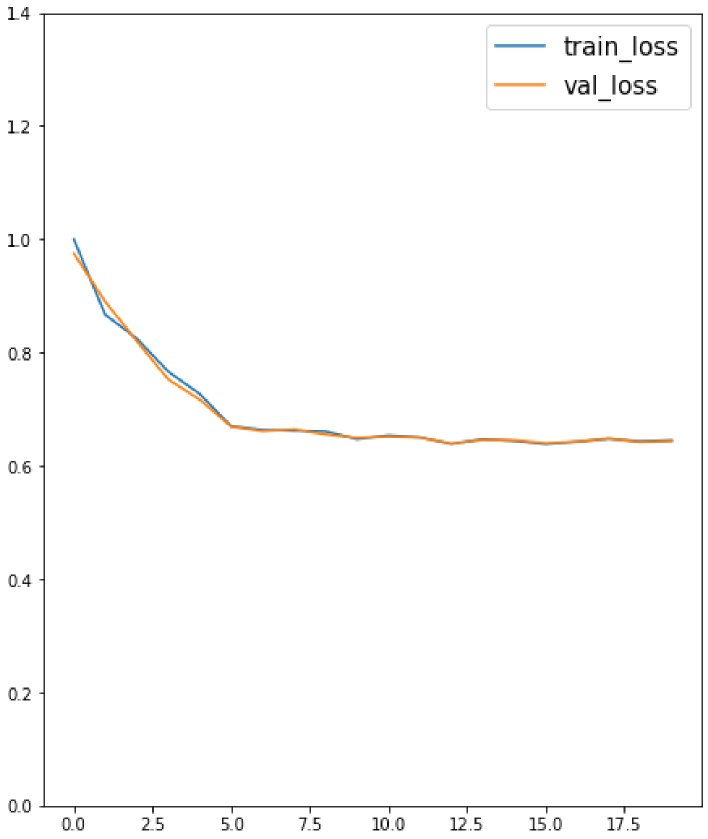
Where t_i $\coloneqq$ the basis vector $e_i \in \mathbb{R}^n$ if the label of x is i.

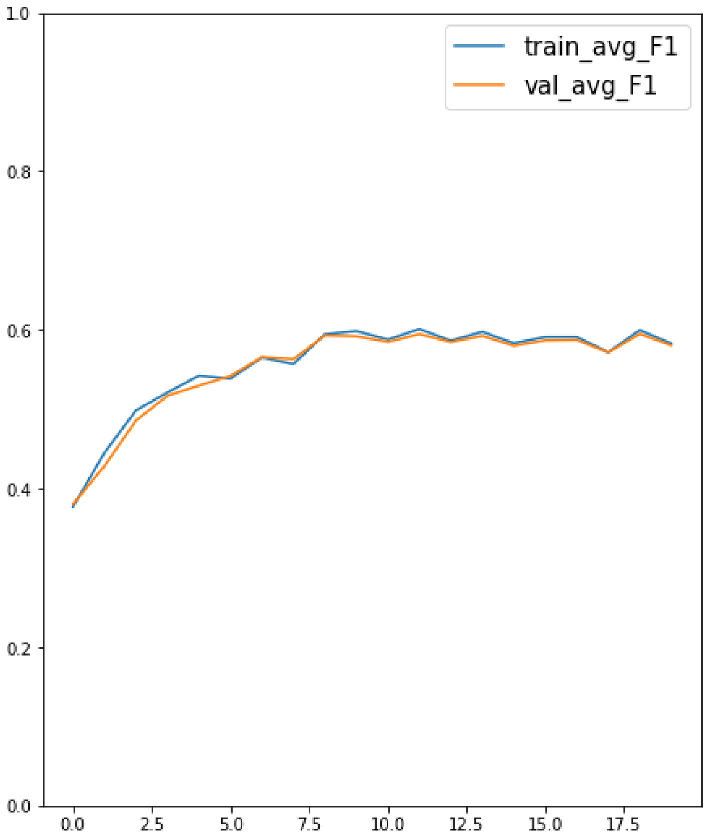


Source: https://www.researchgate.net/figure/The-architecture-of-ResNet50-and-deep-learning-model-flowchart-a-b-Architecture-of_fig1_334767096

The First Task - Results

	Precision (%)	Recall (%)	F1 (%)
Class 1 (NIL)	86.14	76.37	80.29
Class 2 (Moderate)	64.51	92.76	75.54
Class 3 (Severe)	66.63	14.63	22.64





The Second Task - Overview

Model Specifications

- ResNet50 Model (same as Task 1)
- Uses LDAM loss function

$$\mathcal{L}_{\text{LDAM}}((x, y); f) = -\log \frac{e^{z_y - \Delta_y}}{e^{z_y - \Delta_y} + \sum_{j \neq y} e^{z_j}}$$

$$\text{where } \Delta_j = \frac{C}{n_j^{1/4}} \text{ for } j \in \{1, \dots, k\}$$

```python
def ldam(out, label, C, n_j, drw_active):
  deltas = 1/n_j # actually = 1/(n_j)^0.25

  # for a given input x with corresponding label y and model-generated output
  # which is a vector (p_0, p_1, p_2) where p_i denotes the probability the
  # model believes x belongs to class i
  # we need to extract the necessary values into a B*1 vector z_y. Example:
  # (0.4, 0.32, 0.28) | 1
  # (0.8, 0.15, 0.05) | 2
  # (0.2, 0.6, 0.2) | 1

  z_y = out[range(out.shape[0]), label]
  # returns vector of size B*1
  # corresponding to z_y of each element x in the current batch

  delta_y = torch.zeros((label.shape[0],)).cuda() # let delta_y be B*1
  for i in range(3): # since k=3. Flexible if k changes
    delta_y += torch.where(label+1==i+1, label+1, 0)*deltas[i]/(i+1)
  delta_y_copy = delta_y.clone() # used for DRW
  delta_y *= C
  e_zyminusdeltay = torch.exp(z_y-delta_y)

  # since each value of label is in {0, 1, 2}
  z_j_1 = out[range(out.shape[0]), (label+1)%3]
  z_j_2 = out[range(out.shape[0]), (label+2)%3]
  e_zj_sum = torch.exp(z_j_1) + torch.exp(z_j_2)

  # compute L_LDAM((x,y);f)
  # for all x in the batch. Asked TA and he said it was nat log
  result = -torch.log(e_zyminusdeltay/(e_zyminusdeltay+e_zj_sum))

  # reweight LDAM via DRW if necessary via entrywise tensor multiplication
  renormalize_factor = 0
  if drw_active:
    n_y_reciprocal = torch.pow(delta_y_copy, 4)
    result = result * n_y_reciprocal
    renormalize_factor = torch.sum(n_y_reciprocal).item()

  # take average of the entries to get a real-valued result
  return torch.mean(result), renormalize_factor
```

The Second Task - Overview

n_j: number of samples in training set who have label j, if 591 samples have label 0, 839 have label 1, 324 have label 2

deltas = $[1/(591^{0.25}), 1/(839^{0.25}), 1/(324^{0.25})]$

```
def ldam(out, label, C, n_j, drw_active):
  deltas = 1/n_j # actually = 1/(n_j)^0.25

  # for a given input x with corresponding label y and model-generated output
  # which is a vector (p_0, p_1, p_2) where p_i denotes the probability the
  # model believes x belongs to class i
  # we need to extract the necessary values into a B*1 vector z_y. Example:
  # (0.4, 0.32, 0.28) | 1
  # (0.8, 0.15, 0.05) | 2
  # (0.2, 0.6, 0.2) | 1

  z_y = out[range(out.shape[0]), label]
  # returns vector of size B*1
  # corresponding to z_y of each element x in the current batch

  delta_y = torch.zeros((label.shape[0],)).cuda() # let delta_y be B*1
  for i in range(3): # since k=3. Flexible if k changes
    delta_y += torch.where(label+1==i+1, label+1, 0)*deltas[i]/(i+1)
  delta_y_copy = delta_y.clone() # used for DRW
  delta_y *= C
  e_zyminusdeltay = torch.exp(z_y-delta_y)

  # since each value of label is in {0, 1, 2}
  z_j_1 = out[range(out.shape[0]), (label+1)%3]
  z_j_2 = out[range(out.shape[0]), (label+2)%3]
  e_zj_sum = torch.exp(z_j_1) + torch.exp(z_j_2)

  # compute L_LDAM((x,y);f)
  # for all x in the batch. Asked TA and he said it was nat log
  result = -torch.log(e_zyminusdeltay/(e_zyminusdeltay+e_zj_sum))

  # reweight LDAM via DRW if necessary via entrywise tensor multiplication
  renormalize_factor = 0
  if drw_active:
    n_y_reciprocal = torch.pow(delta_y_copy, 4)
    result = result * n_y_reciprocal
    renormalize_factor = torch.sum(n_y_reciprocal).item()

  # take average of the entries to get a real-valued result
  return torch.mean(result), renormalize_factor
```

The Second Task - Overview

n_j: number of samples in training set who have label j, if 591 samples have label 0, 839 have label 1, 324 have label 2

deltas = $[1/(591^{0.25}), 1/(839^{0.25}), 1/(324^{0.25})]$

$$\mathcal{L}_{\mathrm{LDAM}}((x,y);f) = -\log \frac{e^{z_y-\Delta_y}}{e^{z_y-\Delta_y} + \sum_{j\neq y} e^{z_j}}$$

$$\text{where } \Delta_j = \frac{C}{n_j^{1/4}} \text{ for } j \in \{1,\dots,k\}$$

Consider some input x whose model output is (0.4, 0.32, 0.28)
Let y = 1 be x's corresponding label.
So **z_y** just finds (0.4, 0.32, 0.28)[1] = 0.32
Do this for every input x...

delta_y just finds C/(n_y)^0.25 for every x

```
def ldam(out, label, C, n_j, drw_active):
  deltas = 1/n_j # actually = 1/(n_j)^0.25

  # for a given input x with corresponding label y and model-generated output
  # which is a vector (p_0, p_1, p_2) where p_i denotes the probability the
  # model believes x belongs to class i
  # we need to extract the necessary values into a B*1 vector z_y. Example:
  # (0.4, 0.32, 0.28) | 1
  # (0.8, 0.15, 0.05) | 2
  # (0.2, 0.6, 0.2) | 1

  z_y = out[range(out.shape[0]), label]
  # returns vector of size B*1
  # corresponding to z_y of each element x in the current batch

  delta_y = torch.zeros((label.shape[0],)).cuda() # let delta_y be B*1
  for i in range(3): # since k=3. Flexible if k changes
    delta_y += torch.where(label+1==i+1, label+1, 0)*deltas[i]/(i+1)
  delta_y_copy = delta_y.clone() # used for DRW
  delta_y *= C
  e_zyminusdeltay = torch.exp(z_y-delta_y)

  # since each value of label is in {0, 1, 2}
  z_j_1 = out[range(out.shape[0]), (label+1)%3]
  z_j_2 = out[range(out.shape[0]), (label+2)%3]
  e_zj_sum = torch.exp(z_j_1) + torch.exp(z_j_2)

  # compute L_LDAM((x,y);f)
  # for all x in the batch. Asked TA and he said it was nat log
  result = -torch.log(e_zyminusdeltay/(e_zyminusdeltay+e_zj_sum))

  # reweight LDAM via DRW if necessary via entrywise tensor multiplication
  renormalize_factor = 0
  if drw_active:
    n_y_reciprocal = torch.pow(delta_y_copy, 4)
    result = result * n_y_reciprocal
    renormalize_factor = torch.sum(n_y_reciprocal).item()

  # take average of the entries to get a real-valued result
  return torch.mean(result), renormalize_factor
```

The Second Task - Overview

$$\mathcal{L}_{\text{LDAM}}((x,y);f) = -\log \frac{e^{z_y - \Delta_y}}{e^{z_y-\Delta_y} + \sum_{j\neq y} e^{z_j}}$$

$$\text{where } \Delta_j = \frac{C}{n_j^{1/4}} \text{ for } j \in \{1,\dots,k\}$$

Consider some input x whose model output is (0.4, 0.32, 0.28)
Let y = 1 be x's corresponding label.

e_zj_sum just finds e^(z_j) for every <u>incorrect</u> label for x and adds them
Since only 3 labels: {0, 1, 2}
Then the incorrect labels can always be found by (y+1) mod 3 and (y+2) mod 3

Do this for every input x...

Then just plug in yellow and light blue terms into LDAM function

```python
def ldam(out, label, C, n_j, drw_active):
  deltas = 1/n_j # actually = 1/(n_j)^0.25

  # for a given input x with corresponding label y and model-generated output
  # which is a vector (p_0, p_1, p_2) where p_i denotes the probability the
  # model believes x belongs to class i
  # we need to extract the necessary values into a B*1 vector z_y. Example:
  # (0.4, 0.32, 0.28) | 1
  # (0.8, 0.15, 0.05) | 2
  # (0.2, 0.6, 0.2) | 1

  z_y = out[range(out.shape[0]), label]
  # returns vector of size B*1
  # corresponding to z_y of each element x in the current batch

  delta_y = torch.zeros((label.shape[0],)).cuda() # let delta_y be B*1
  for i in range(3): # since k=3. Flexible if k changes
    delta_y += torch.where(label+1==i+1, label+1, 0)*deltas[i]/(i+1)
  delta_y_copy = delta_y.clone() # used for DRW
  delta_y *= C
  e_zyminusdeltay = torch.exp(z_y-delta_y)

  # since each value of label is in {0, 1, 2}
  z_j_1 = out[range(out.shape[0]), (label+1)%3]
  z_j_2 = out[range(out.shape[0]), (label+2)%3]
  e_zj_sum = torch.exp(z_j_1) + torch.exp(z_j_2)

  # compute L_LDAM((x,y);f)
  # for all x in the batch. Asked TA and he said it was nat log
  result = -torch.log(e_zyminusdeltay/(e_zyminusdeltay+e_zj_sum))

  # reweight LDAM via DRW if necessary via entrywise tensor multiplication
  renormalize_factor = 0
  if drw_active:
    n_y_reciprocal = torch.pow(delta_y_copy, 4)
    result = result * n_y_reciprocal
    renormalize_factor = torch.sum(n_y_reciprocal).item()

  # take average of the entries to get a real-valued result
  return torch.mean(result), renormalize_factor
```

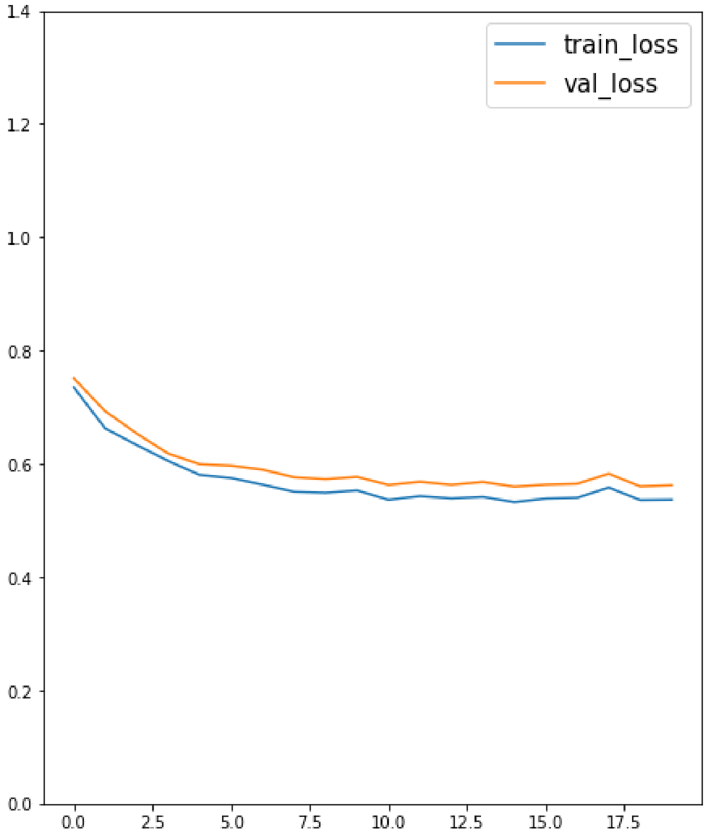
The Second Task - Overview

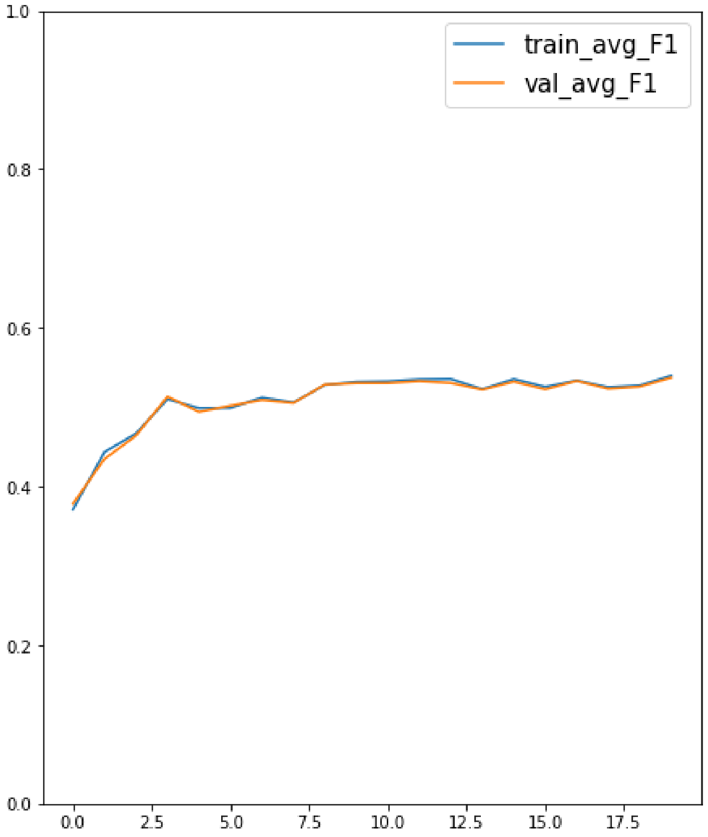
- **Hyperparameters to tweak:**
  - C
  - Learning rate
  - Momentum term
  - alpha (regularization)
  - Batch size
- <u>Calculating the optimal set of hyperparameters is infeasible</u>
- **Solution: exact grid search**
  - Try every possible combination of parameters
  - Too many combinations...
- Instead, opt to fix some values: use industry standard for batch size (64) momentum (0.9) and learning rate (0.001)
- alpha = (0.001, **0.01**, 0.1, 1), C = (-2, **-0.25**, 0.25, 2)

```
def ldam(out, label, C, n_j, drw_active):
  deltas = 1/n_j # actually = 1/(n_j)^0.25

  # for a given input x with corresponding label y and model-generated output
  # which is a vector (p_0, p_1, p_2) where p_i denotes the probability the
  # model believes x belongs to class i
  # we need to extract the necessary values into a B*1 vector z_y. Example:
  # (0.4, 0.32, 0.28) | 1
  # (0.8, 0.15, 0.05) | 2
  # (0.2, 0.6, 0.2) | 1

  z_y = out[range(out.shape[0]), label]
  # returns vector of size B*1
  # corresponding to z_y of each element x in the current batch

  delta_y = torch.zeros((label.shape[0],)).cuda() # let delta_y be B*1
  for i in range(3): # since k=3. Flexible if k changes
    delta_y += torch.where(label+1==i+1, label+1, 0)*deltas[i]/(i+1)
  delta_y_copy = delta_y.clone() # used for DRW
  delta_y *= C
  e_zyminusdeltay = torch.exp(z_y-delta_y)

  # since each value of label is in {0, 1, 2}
  z_j_1 = out[range(out.shape[0]), (label+1)%3]
  z_j_2 = out[range(out.shape[0]), (label+2)%3]
  e_zj_sum = torch.exp(z_j_1) + torch.exp(z_j_2)

  # compute L_LDAM((x,y);f)
  # for all x in the batch. Asked TA and he said it was nat log
  result = -torch.log(e_zyminusdeltay/(e_zyminusdeltay+e_zj_sum))

  # reweight LDAM via DRW if necessary via entrywise tensor multiplication
  renormalize_factor = 0
  if drw_active:
    n_y_reciprocal = torch.pow(delta_y_copy, 4)
    result = result * n_y_reciprocal
    renormalize_factor = torch.sum(n_y_reciprocal).item()

  # take average of the entries to get a real-valued result
  return torch.mean(result), renormalize_factor
```

The Second Task - Results

	Precision (%)	Recall (%)	F1 (%)
Class 1 (NIL)	87.51	67.84	75.45
Class 2 (Moderate)	60.19	94.66	72.90
Class 3 (Severe)	52.18	7.56	12.80



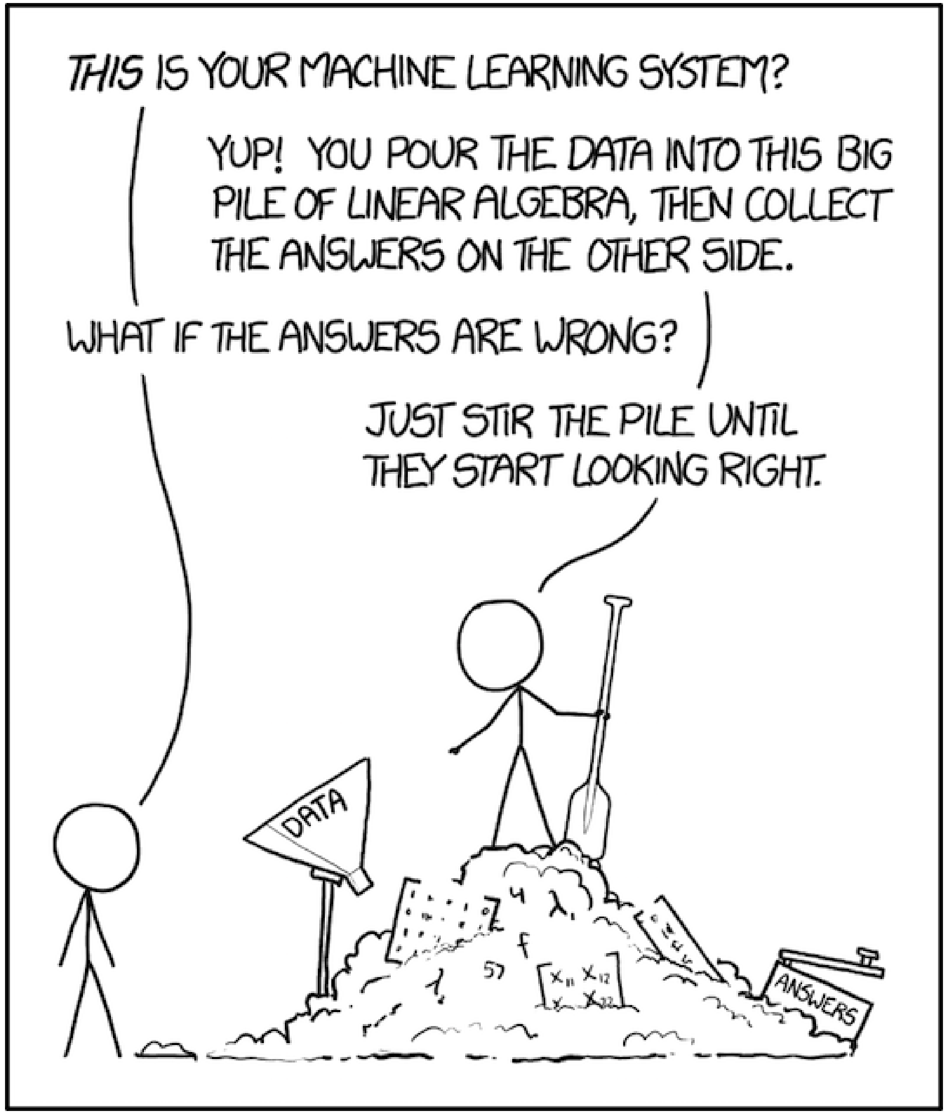


The Final Model

Issues With the Other Models

- Overfit to majority classes in the training set
- Doesn’t perform well on the third class
- Lacking in recall / F1

Solution?
Try different models with different loss functions and see which one meets our design goals.



Source: https://xkcd.com/1838/

The Different Models We Tried

1. ResNet50 with Recall Loss
2. Wide ResNet50 2 with Recall Loss
3. ResNet50 with Focal Loss

What is Recall Loss?
What is Focal Loss?

What can these models do better than the first two models?

What's the difference between Wide ResNet50 2 and ResNet50?

$$L(x)_{Recall} = -\log\left(\frac{w_i e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}\right)$$

Where w_i is the false negative rate of the model on class i on the current batch.

$$FL(p_t) = -(1-p_t)^{\gamma}\log(p_t)$$

Where $p_t = \begin{cases} p, & if\ y = 1 \\ 1-p, & otherwise \end{cases}$

And p is the models estimate for the class with label y, the class identifier.

Contender Model Results

1

ResNet50 with Recall Loss

	Precision (%)
NIL	87.08
MOD	68.34
SEV	46.03
	Recall (%)
NIL	74.08
MOD	75.15
SEV	54.50
	F1 (%)
NIL	79.60
MOD	71.03
SEV	48.32

2

Wide ResNet50 2 with Recall Loss

	Precision (%)
NIL	87.30
MOD	65.73
SEV	53.49
	Recall (%)
NIL	72.09
MOD	77.46
SEV	57.72
	F1 (%)
NIL	78.04
MOD	70.47
SEV	54.85

3

ResNet50 with Focal Loss

	Precision (%)
NIL	88.01
MOD	58.86
SEV	62.98
	Recall (%)
NIL	68.64
MOD	89.03
SEV	21.13
	F1 (%)
NIL	76.55
MOD	70.18
SEV	30.20

Contender Model Results

1

ResNet50 with Recall Loss

	Precision (%)
NIL	87.08
MOD	68.34
SEV	46.03
	Recall (%)
NIL	74.08
MOD	75.15
SEV	54.50
	F1 (%)
NIL	79.60
MOD	71.03
SEV	48.32

2

Wide ResNet50 2 with Recall Loss

	Precision (%)
NIL	87.30
MOD	65.73
SEV	53.49
	Recall (%
NIL	72.09
MOD	77.46
SEV	57.72
	F1 (%)
NIL	78.04
MOD	70.47
SEV	54.85

3

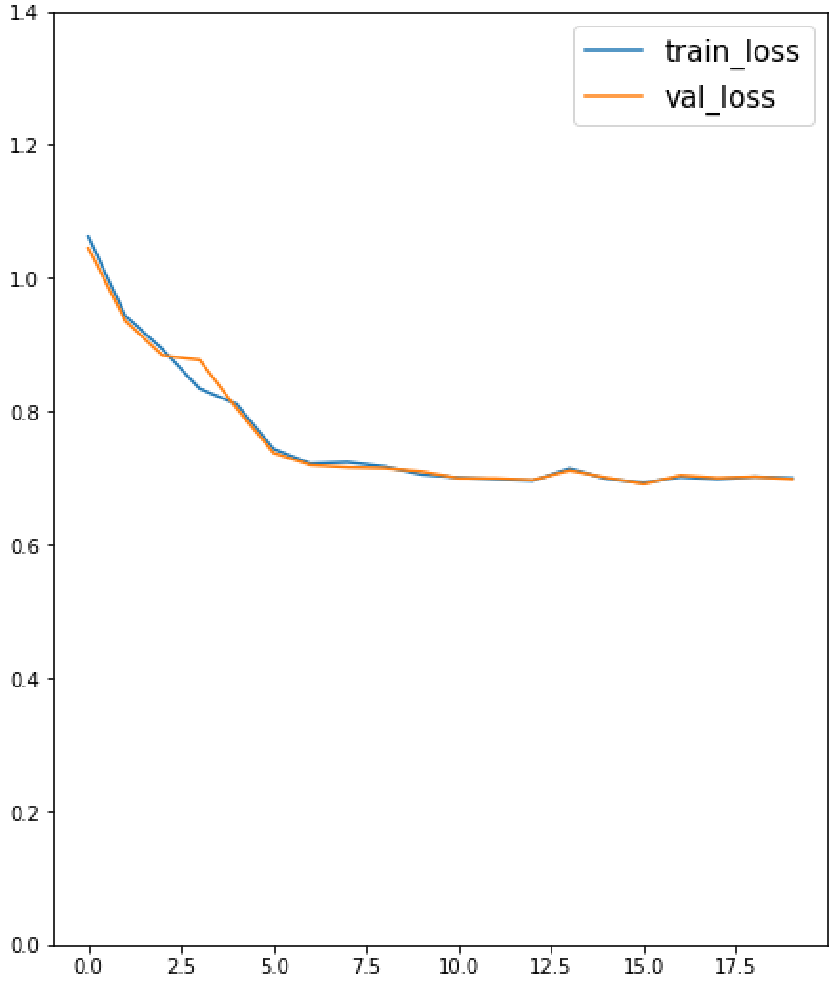
ResNet50 with Focal Loss

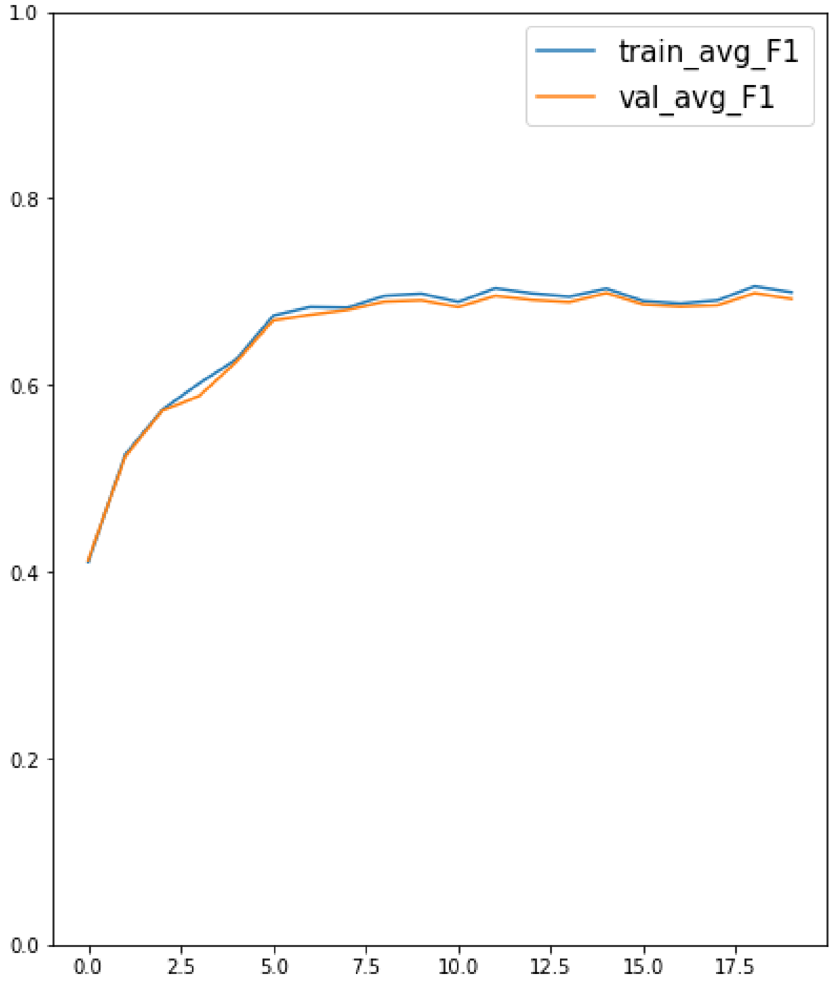
	Precision (%)
NIL	88.01
MOD	58.86
SEV	62.98
	Recall (%)
NIL	68.64
MOD	89.03
SEV	21.13
	F1 (%)
NIL	76.55
MOD	70.18
SEV	30.20

Retrain the Chosen Model

- Wide ResNet50 2 with Recall Loss

	Precision (%)	Recall (%)	F1 (%)
NIL	88.53	74.52	80.28
Moderate	69.91	79.44	73.65
Severe	53.59	59.79	55.58





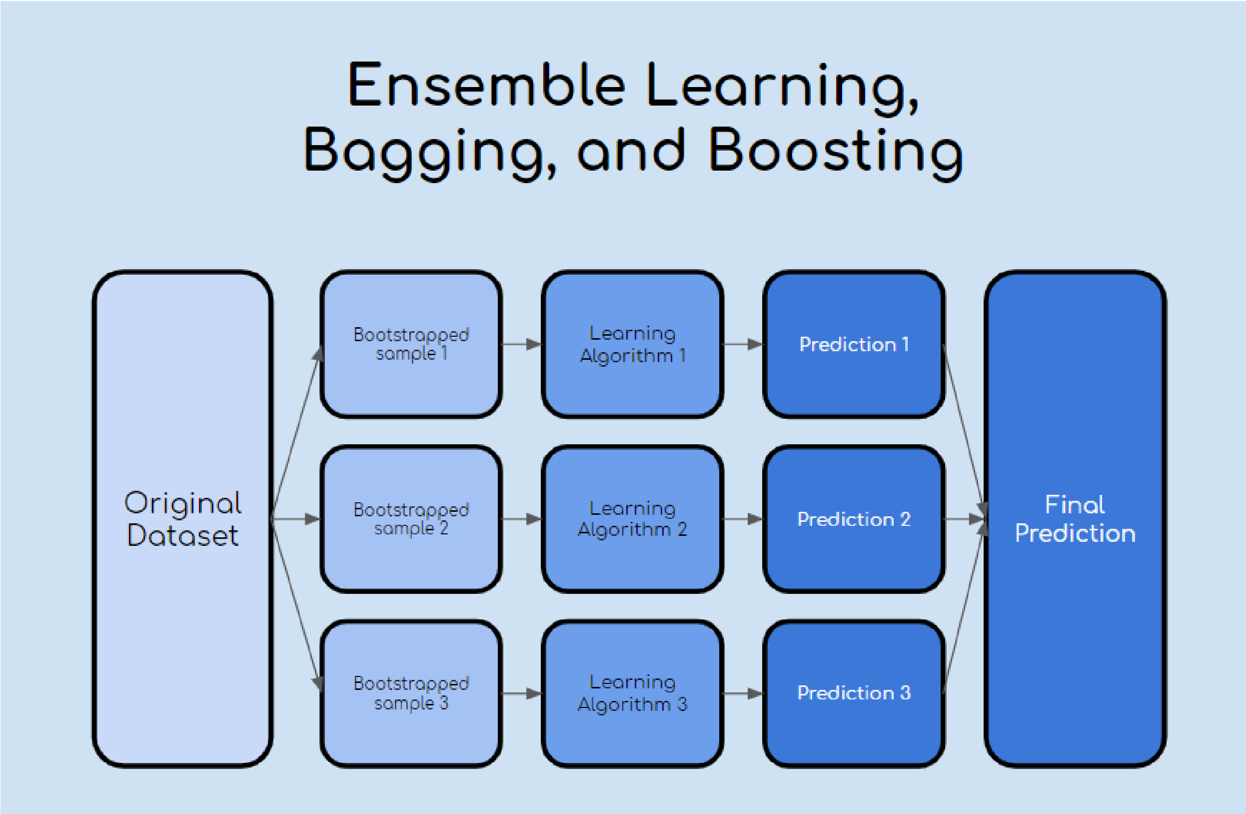
Conclusion

The Final Model – What Could've Been Done Differently

- Ensemble techniques
  - Bootstrap aggregating (Bagging)

- More Models
  - DenseNet, AlexNet, etc.
  - In general, CNNs are not rotation invariant
  - Customised model with max pooling?

- Domain-specific knowledge



Related Works and Information

- Recall Loss: https://openreview.net/pdf?id=SlprFTIQP3
- Focal Loss: https://arxiv.org/pdf/1708.02002.pdf

🎅 Thank You For Listening! 🎅

Any Questions?