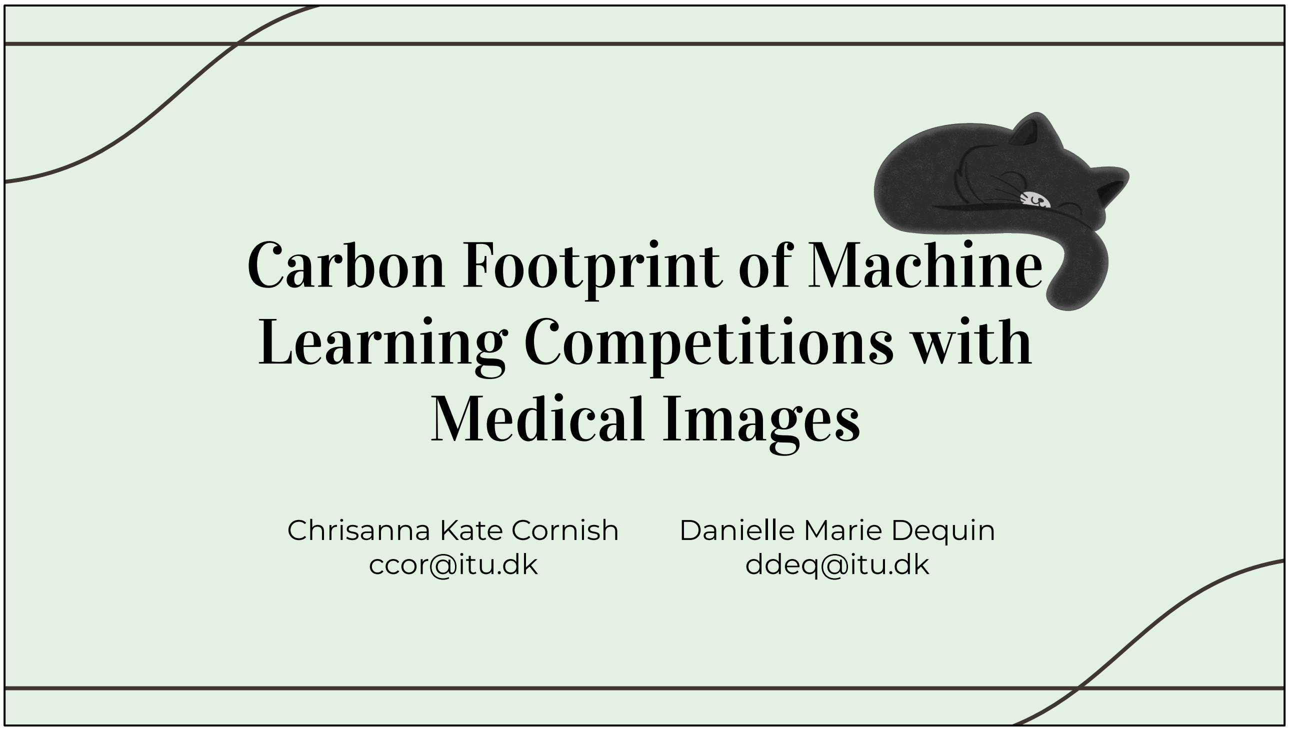
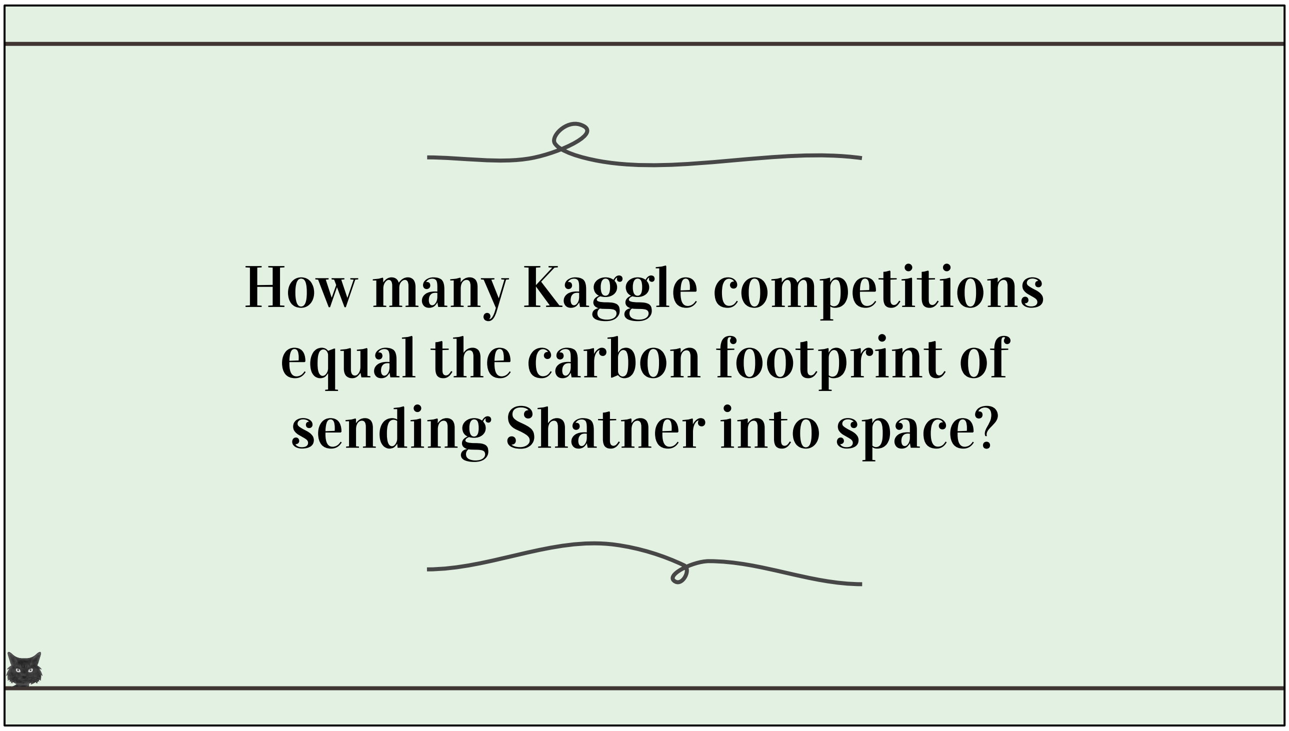


Carbon Footprint of Machine Learning Competitions with Medical Images

Chrisanna Kate Cornish
ccor@itu.dk

Danielle Marie Dequin
ddeq@itu.dk

Hi I'm Danielle and this is my colleague Sanna and we are here to present our bachelor's project on the Carbon Footprint of Machine learning Competitions with Medical Images



How many Kaggle competitions equal the carbon footprint of sending Shatner into space?

We started wondering… how many machine learning Kaggle competitions does it take to generate the same carbon footprint as sending William Shatner into Space?

If you don’t know, around 18 months ago, the 90 year old star trek actor took a short trip to space in the Blue Origin craft.

Despite the rocket itself emitting little carbon, the carbon footprint of this jolly was estimated to be around 75 tonnes per person. When we consider that over a billion people on the planet today will be responsible for less than 1 tonne in their lifetime, this seems rather high.

So what’s it got to do with machine learning, or Kaggle?

Machine learning is expensive…

…and getting more so.

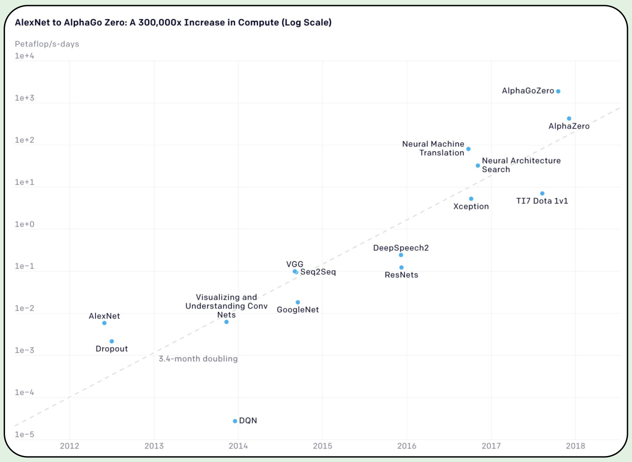


Figure from https://openai.com/blog/ai-and-compute/

Well not a lot, but it does give us something we can picture. A literal spacerocket.

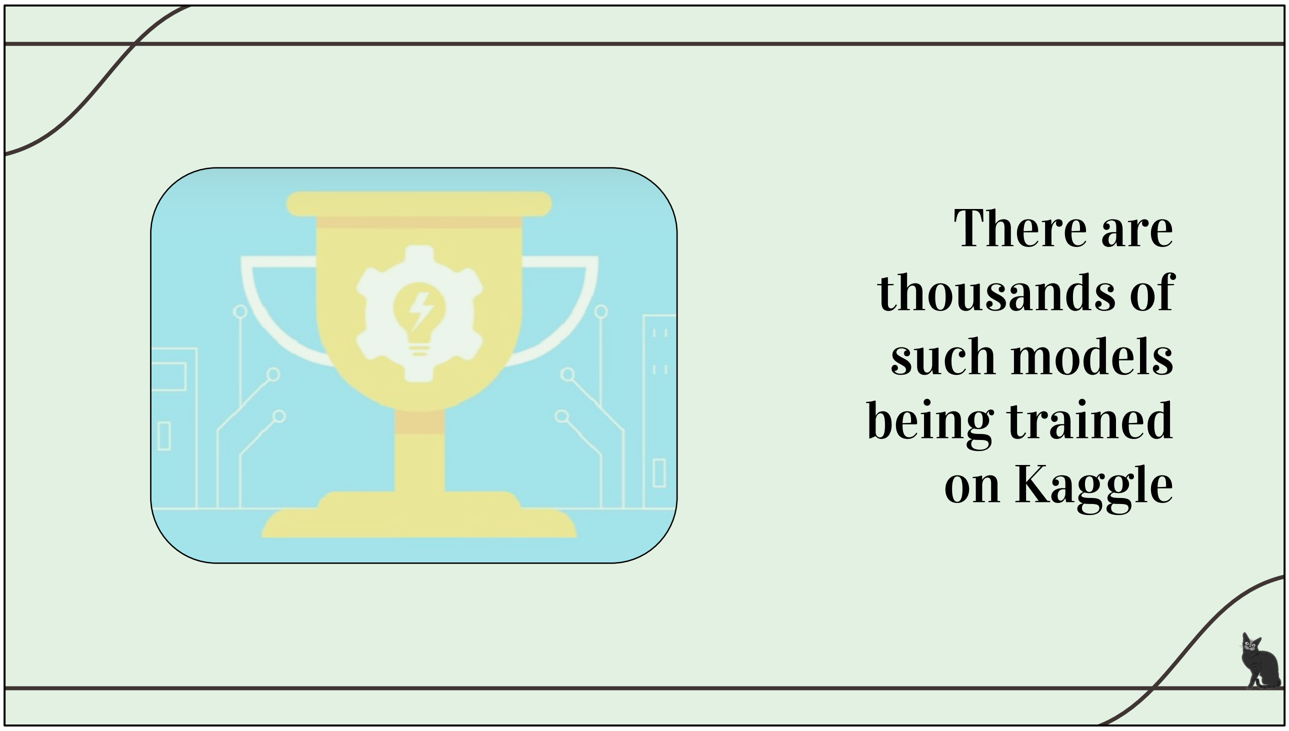
As you probably know, training machine learning models can be computationally expensive, causing a high energy consumption.
Especially large, deep learning models that are trained on huge datasets.

In the last 10 years, there has been a massive jump in the computational power required to train models, roughly doubling every 3-4 months, where previously (and not shown here), this trend doubled approximately every 2 years.
This is impressive. However, even though hardware is more energy-efficient than ever, the sheer amount of processing needed to handle this much data is costly.

For example, we found research that shows the carbon footprint of a Neural Architecture Search, known for being computationally intensive, can be over 284 tonnes CO2 eq. Far more than a space flight.

—--

NOTE: Look up petaflops
A petaflop/s-day (pfs-day) consists of performing 10^{15} neural net operations per second for one day, or a total of about 10^{20} operations.



There are thousands of such models being trained on Kaggle

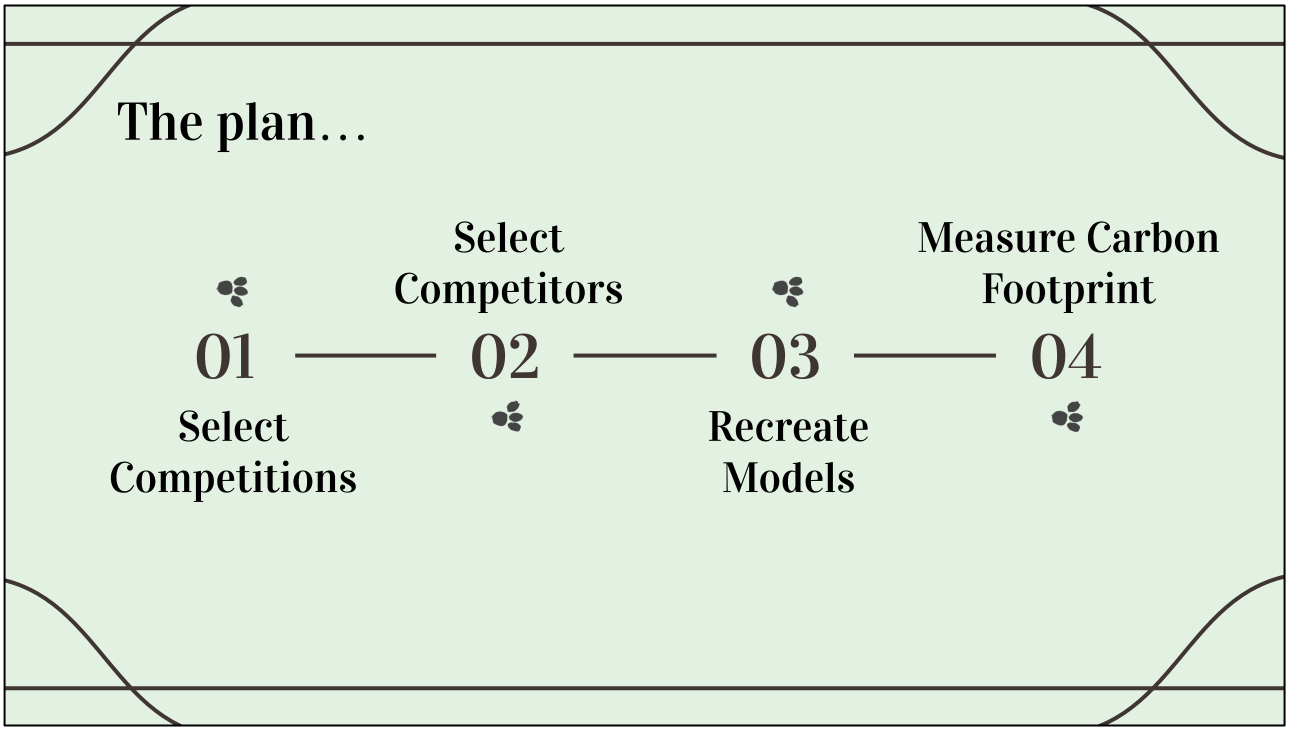
Kaggle is an online community and platform for machine learning competitions where large numbers of models are being trained by competitors around the world.

It's an accessible 'playground' with small training sets and walkthroughs for beginners, and as well as competitions aimed at solving real-world problems.
As of April, Kaggle was hosting 108 image machine learning competitions, most of which have many thousands of submissions.

Recent research has looked into 8 particular Kaggle competitions, finding that often the winning method does not make significant improvements compared to the evaluation noise.

With the cost of machine learning being high, and such a high volume of models being trained on Kaggle with winning methods not making significant improvements, we wanted to know: What is the carbon footprint of a Kaggle competition?

So how did we do this…



The plan…

01 Select Competitions — 02 Select Competitors — 03 Recreate Models — 04 Measure Carbon Footprint

Originally, we planned to choose a couple medical image competitions, and then select a couple competitors with available code from the top 10.
We would then recreate their models, and measure the carbon footprint of training them.
Using those results, we could estimate the carbon footprint of medical image competitions on kaggle by looking at the total number of competitors.

Sounds easy…

We decided we would start with one competition, work it through to the end to iron out any issues in our workflow, and then move onto the next.

Category “Code”

Recent

+ Medical Imaging =

SIIM-FISABIO-RSNA COVID-19 competition

To begin we had to select a competition. We were specifically interested in medical imaging competitions, due to the often high requirements needed to train a model.

Our initial idea was to use the Data Bowl 2017, a lung cancer detection competition. However we quickly ran into problems. Firstly, few teams had code still available, if they ever did. Secondly, Kaggle was no longer hosting the dataset, and it was difficult to locate elsewhere.

To avoid this, we decided to narrow it down to “code” competitions. Entrants to this category of competition are required to submit a notebook on Kaggle, which should make it easier to find their code to evaluate.
We also considered that this would help us to estimate how many times they had trained a model as Kaggle records each run as a submission.
Additionally, there are hardware and runtime restrictions for code competitions, which should balance the entries, making it easier to evaluate a few entries and generalize to the whole competition.

We also focussed on more recent competitions, as it seemed that both data and code were more likely to still be available.

This led us to the SIIM FISABIO-RSNA COVID-19 competition, where competitors aimed to identify and localize COVID-19 abnormalities on chest radiographs.

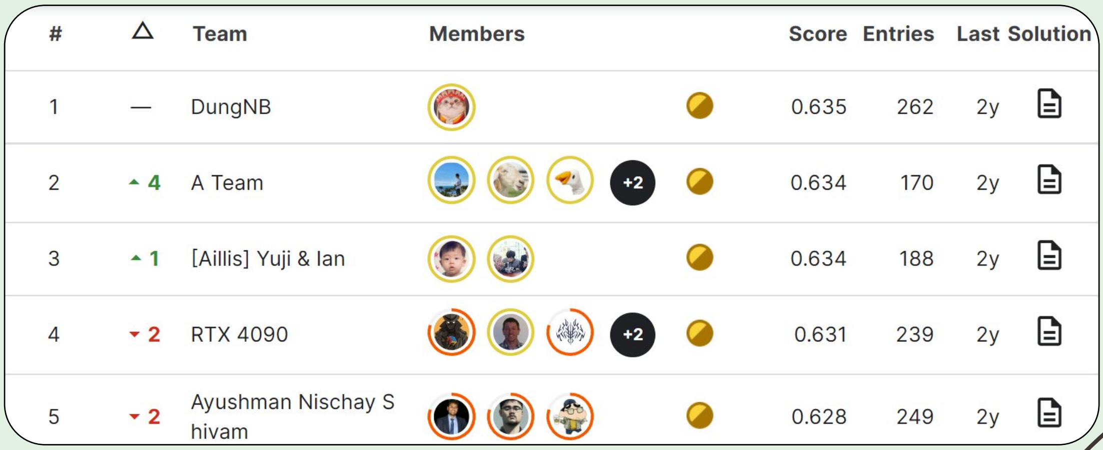
Now we needed to choose a submitting team to evaluate…

—---

NOTES:

As the leading healthcare organization in their field, the Society for Imaging Informatics in Medicine (**SIIM**)'s mission is to advance medical imaging informatics through education, research, and innovation. SIIM has partnered with the Foundation for the Promotion of Health and Biomedical Research of Valencia Region (**FISABIO**), Medical Imaging Databank of the Valencia Region (**BIMCV**) and the Radiological Society of North America (**RSNA**) for this competition.

Competitor Selection

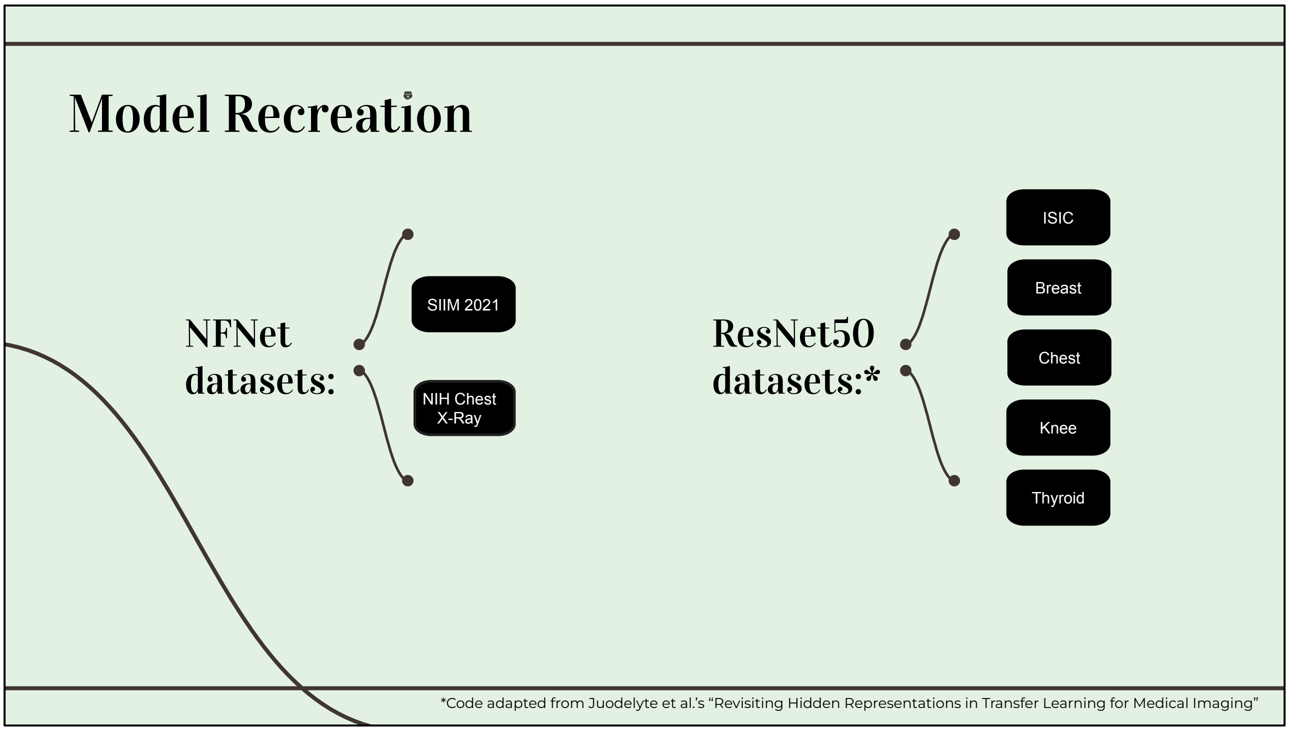


#	△	Team	Members		Score	Entries	Last	Solution
1	—	DungNB			0.635	262	2y	
2	▲ 4	A Team	+2		0.634	170	2y	
3	▲ 1	[Aillis] Yuji & Ian			0.634	188	2y	
4	▼ 2	RTX 4090	+2		0.631	239	2y	
5	▼ 2	Ayushman Nischay S hivam			0.628	249	2y	

We went to the private leaderboard.
The private leaderboard determines the winner, based on evaluation on a private test set of the data.

We realised that all of the high ranking teams had actually completed substantial amounts of work offline and used supplementary datasets.
This seemed to discredit our assumption that “code” competitions would be balanced.

The first placed team used supplementary data that was no longer accessible, so we moved to the second place team.



Model Recreation

NFNet datasets:

- SIIM 2021
- NIH Chest X-Ray

ResNet50 datasets:*

- ISIC
- Breast
- Chest
- Knee
- Thyroid

*Code adapted from Juodelyte et al.'s "Revisiting Hidden Representations in Transfer Learning for Medical Imaging"

And here is where it really started going wrong.

We needed to recreate their training procedure in order to evaluate it. This process took a lot of time, and even so we were not able to get this team's entire pipeline to run.
Faced with running out of time, we needed to find additional data to supplement our estimations…

We decided one way would be to run some other, similar, medical image models, with more accessible, and reproducible code (and where the author was available should we run into issues).
We could then make the assumption that these models could stand as proxies for the models that we were not able to get running.
For this, we 'borrowed' the code from the research paper 'Revisiting Hidden Representations in Transfer Learning for Medical Imaging'.

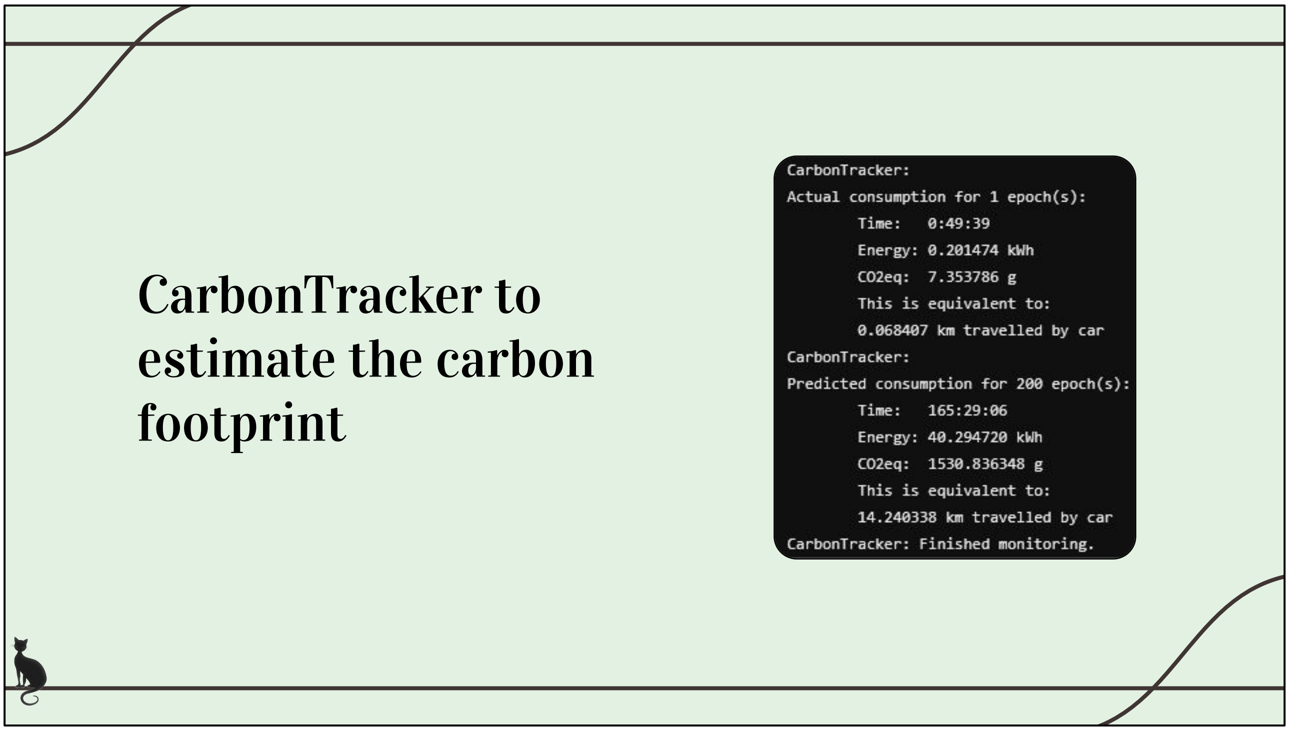
In the SIIM-FISABIO-RSNA COVID-19 competition some of the high scoring teams were using either a RESNET or NFNet architecture.
The second place team that we evaluated used NFNet in their pipeline, and trained on the SIIM 2021 and NIH Chest X-ray datasets.
The supplementary models we evaluated were based on ResNet50, with separate training on 5 different datasets.
We ran training on each dataset twice, with 2 different parameters (freeze True/False), giving us a total of 10 additional models to evaluate.

—---

NOTE:The SIIM 2021 dataset they use is a resized version of the standard dataset for the competition

NFNets are a family of modified ResNets that achieves competitive accuracies without batch normalization.

Other teams also used EfficientNet, Yolo models,



CarbonTracker to estimate the carbon footprint

```
CarbonTracker:
Actual consumption for 1 epoch(s):
        Time:   0:49:39
        Energy: 0.201474 kWh
        CO2eq:  7.353786 g
        This is equivalent to:
        0.068407 km travelled by car
CarbonTracker:
Predicted consumption for 200 epoch(s):
        Time:   165:29:06
        Energy: 40.294720 kWh
        CO2eq:  1530.836348 g
        This is equivalent to:
        14.240338 km travelled by car
CarbonTracker: Finished monitoring.
```

Once we had runnable code with trainable models, we needed to choose a tool to measure the carbon footprint.

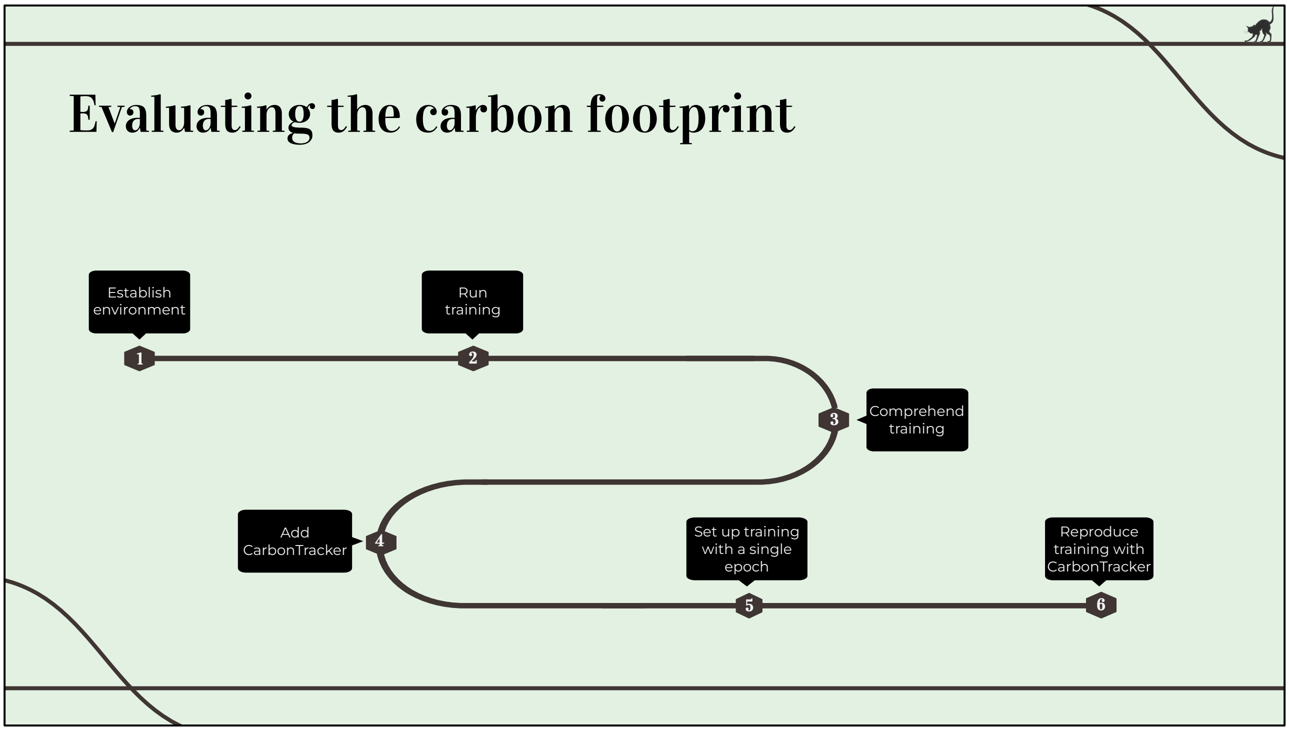
Neighbor researchers at KU developed a tool called CarbonTracker. The values tracked, and then predicted are typical for the field, which would give us a good comparison.
Also, if we happened to run into any problems, we could potentially ask the developers.

Another feature we found important was that CarbonTracker is able to run on a single training iteration and then extrapolate the cost of the full training.
This minimizes our own carbon footprint and training time required.

Once we decided upon CarbonTracker, we were ready to start evaluating some models.

—---

NOTE:
The values tracked are training time (HH:MM:SS), energy usage (kWh), CO2 equivalent (g), and distance traveled by car (km).

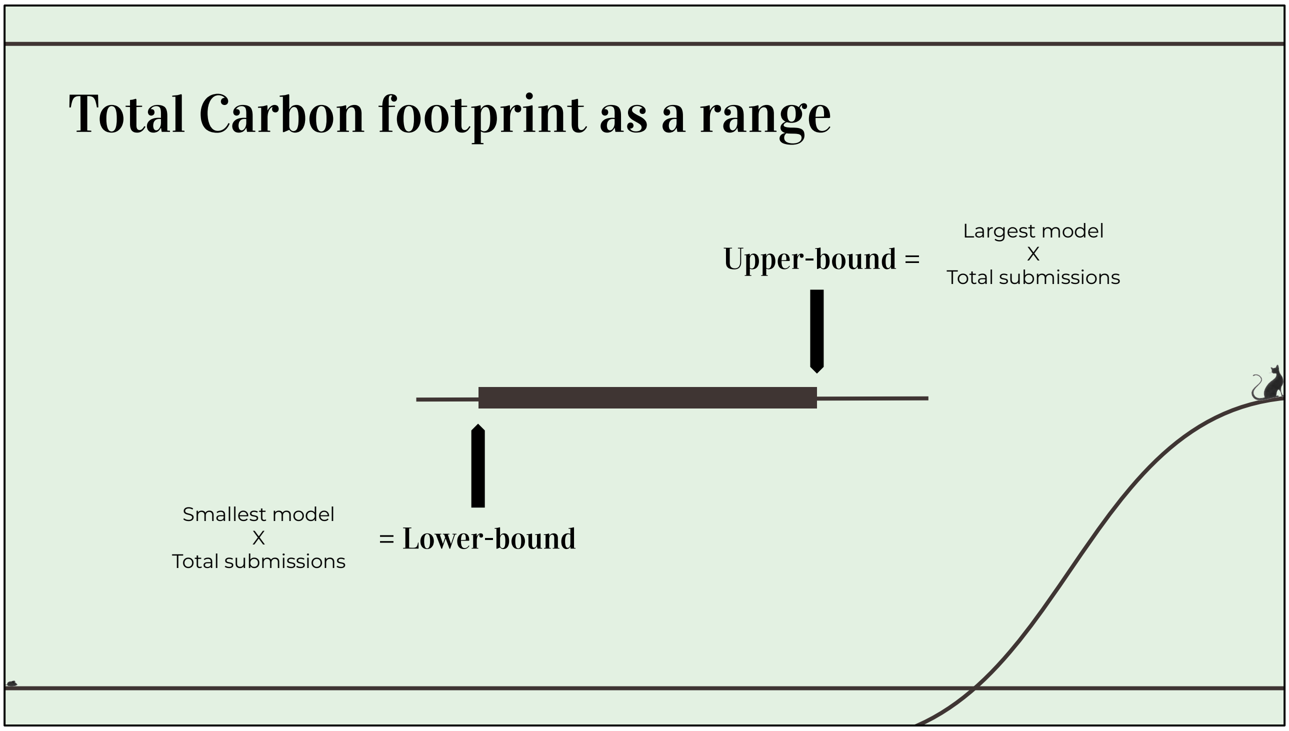


Evaluating the carbon footprint

To do so, we followed a 6-step procedure

1st, we established the necessary environment to run the code. This required 2 separate environments, one for the NFNet model and another for the ResNet models.
2nd, we checked that training runs without CarbonTracker added and that no issues occur. Debug as needed.
3rd, commented out training and added print statements to further understand epochs, etc needed for training. This is essential for setting the maximum number of epochs for CarbonTracker to estimate the total cost without having to run the entire training.
4th, we added CarbonTracker to the code and ran a short test with a 45 minute time limit. Any issues should appear within this time to verify if CarbonTracker was set up correctly.
5th step, we set up training to run with a single epoch. (with necessary break statements)
Lastly, we reproduced a single epoch of the model's training procedure with CarbonTracker added to get the estimated full training cost.

We could then use these results to estimate the carbon footprint of the entire competition.



Total Carbon footprint as a range

Upper-bound = Largest model X Total submissions

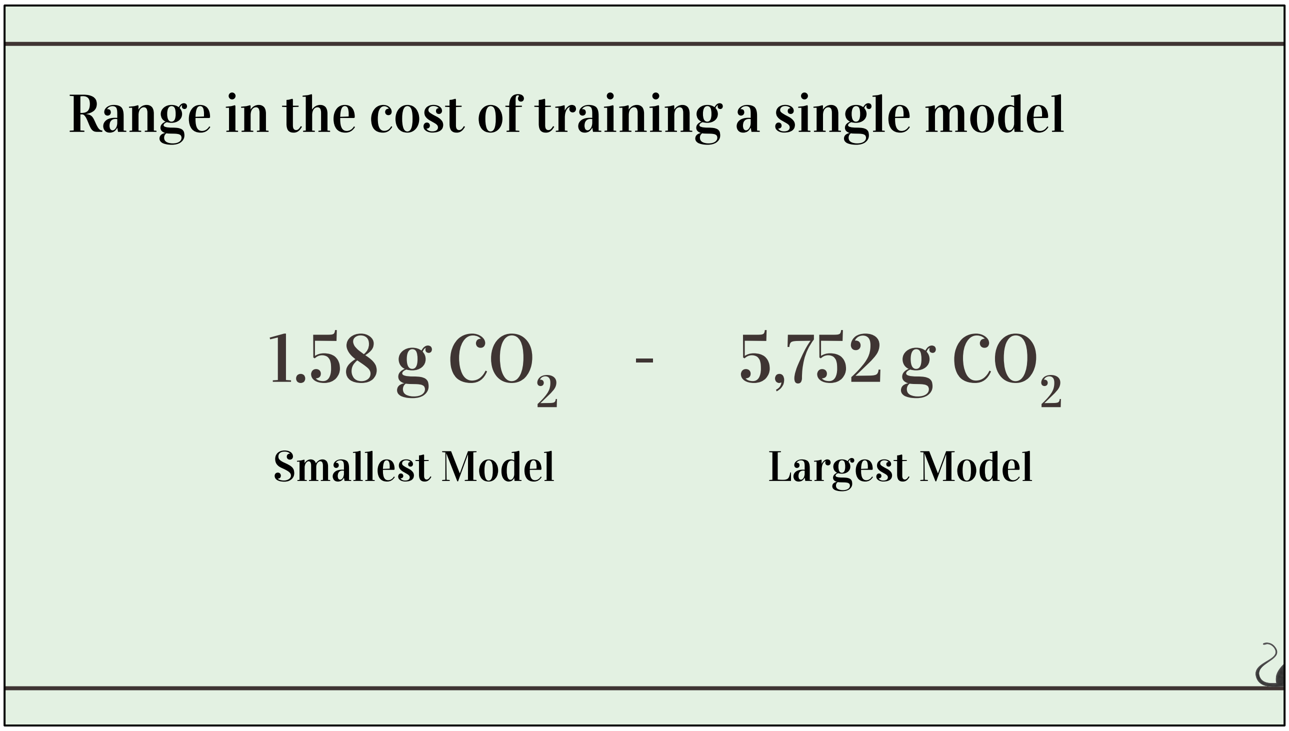
Smallest model X Total submissions = Lower-bound

To do so we calculated a lower and upper bound cost.

The lower bound is based on the model with the smallest carbon footprint out of those we measure. The upper bound is based the model with the largest carbon footprint.

We then needed to multiply both figures (the smallest and largest model costs) by the total number of submissions in the competition.
We also were inspired by other research to not include teams who had submitted less than 3 times, and therefore perhaps did not fully participate.

So what do these results look like…



Range in the cost of training a single model

1.58 g CO_2 - 5,752 g CO_2

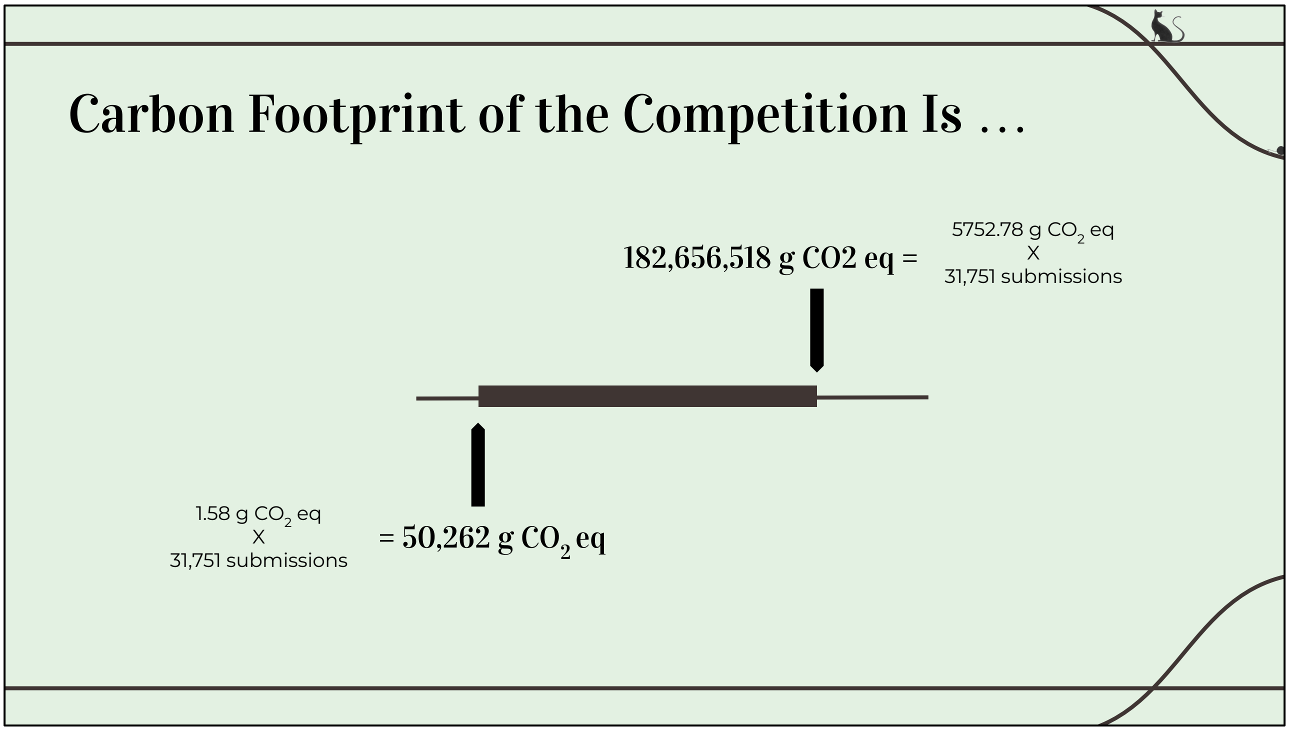
Smallest Model — Largest Model

First we found variation in the carbon footprint of training individual models.
The smallest model cost around 1.6 g CO_2 to train*.
This model happened to be one of the ResNet models.

The largest model was the ensemble of NFNet models submitted by the 2nd place team, which cost 5,752 g CO_2.
This is equivalent to over 53 km driven by a car.

We then use these values to calculate the cost of the whole competition…

—---
NOTE: *the equivalent to 0.02km driven by a car



Carbon Footprint of the Competition Is …

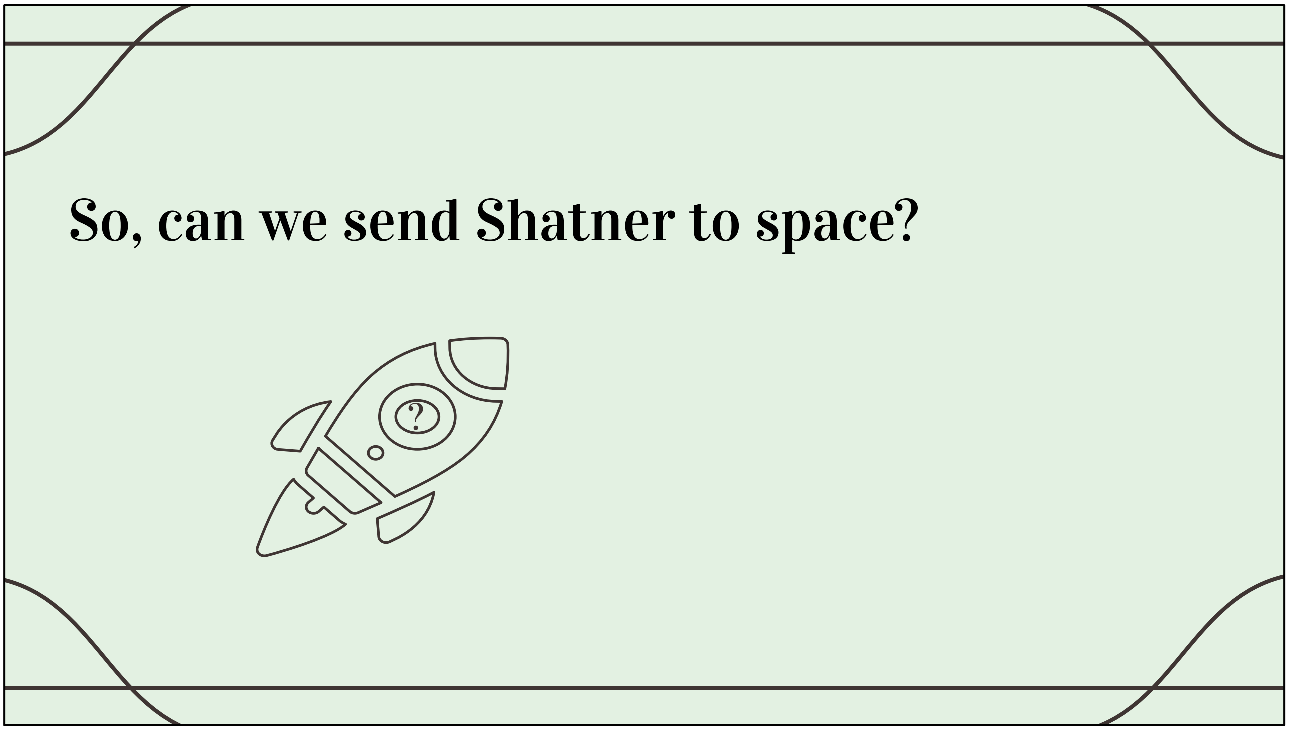
182,656,518 g CO2 eq = 5752.78 g CO_2 eq X 31,751 submissions

1.58 g CO_2 eq X 31,751 submissions = 50,262 g CO_2 eq

The lower bound cost of the competition we estimate to be over 50kg CO2, which is equivalent to driving 476 km by car.
The upper bound is over 182 tonnes CO2, equivalent to over 1,698,000 km driven by a car.

That sounds like a lot to me. But let's think of it another way…

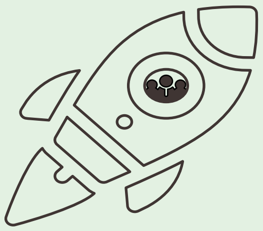
—---
NOTE: Total submissions = 32,307 for all 1,305 teams
Total submissions with groups having >= 3 submissions = 31,751.



So, can we send Shatner to space?

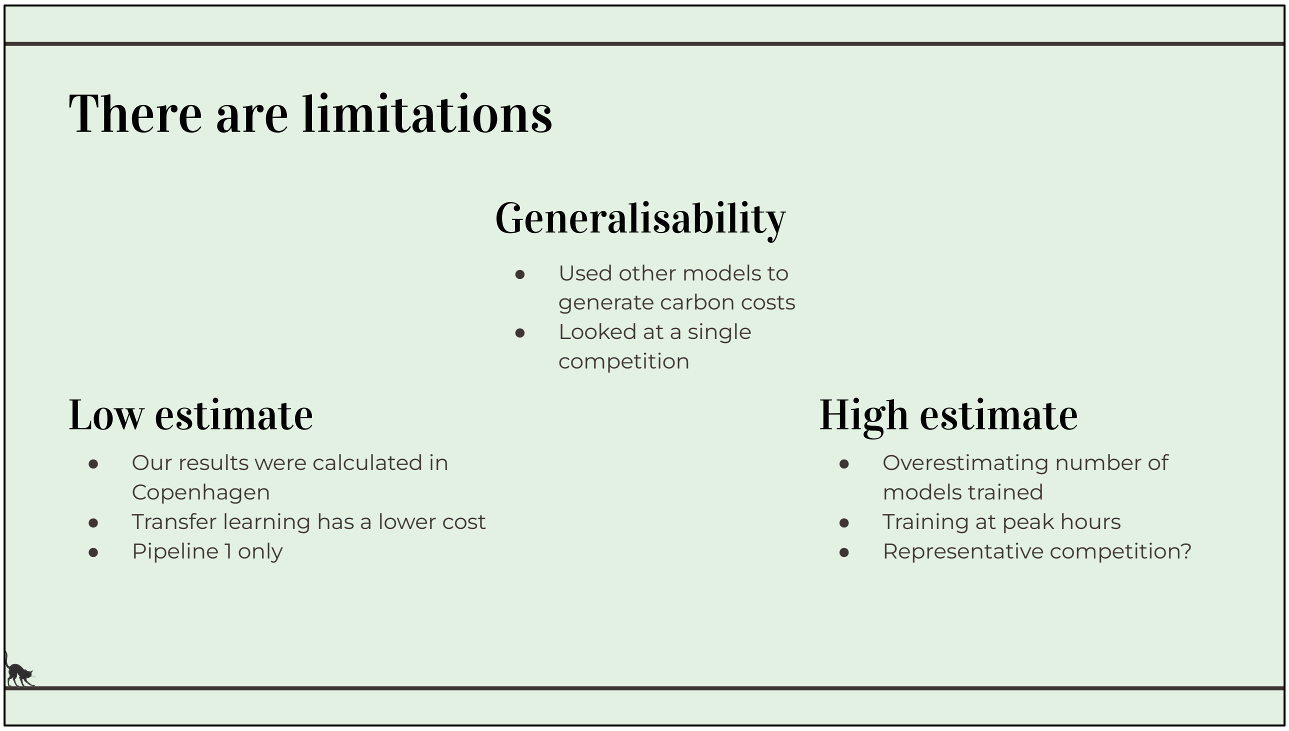
So, can we sent Shatner to space?

So, can we send Shatner to space?



Yes!

Along with 1 other, according to the upper bound estimation of this competition.



There are limitations

Generalisability

- Used other models to generate carbon costs
- Looked at a single competition

Low estimate

- Our results were calculated in Copenhagen
- Transfer learning has a lower cost
- Pipeline 1 only

High estimate

- Overestimating number of models trained
- Training at peak hours
- Representative competition?

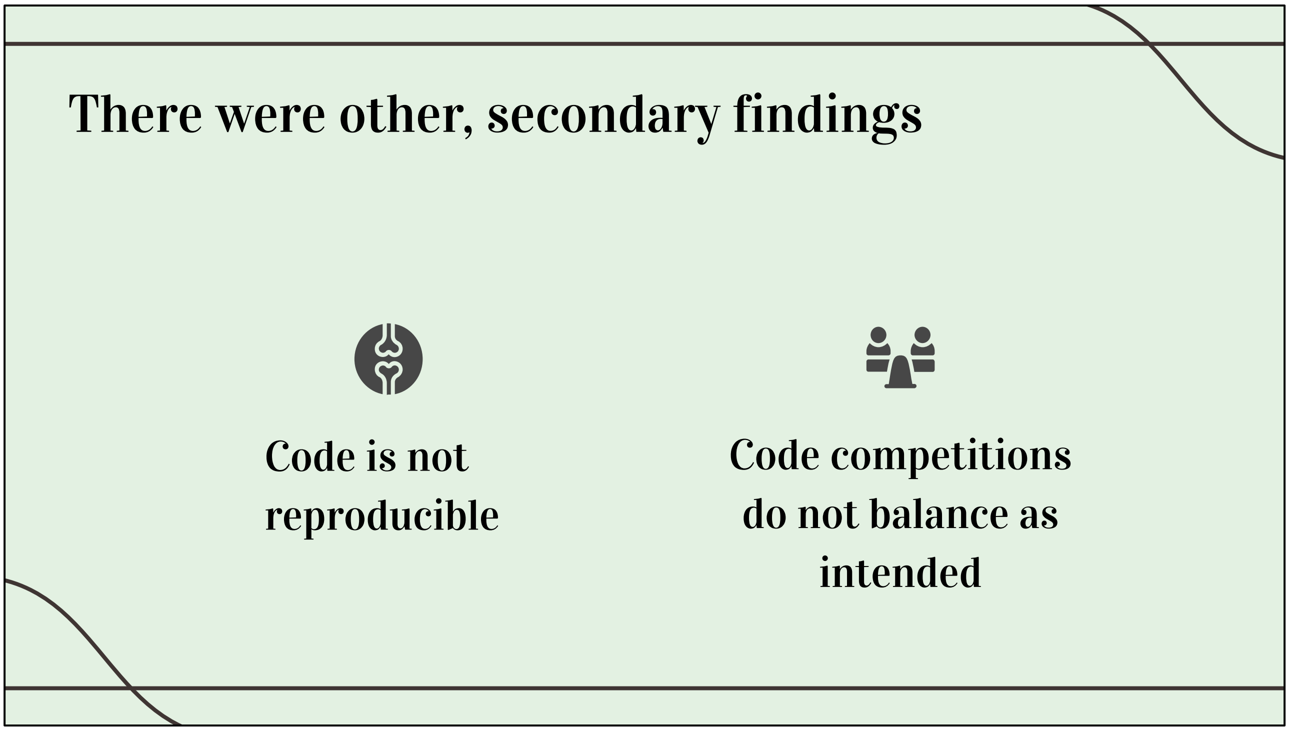
However, there are limitations to our methods.

First of all, we have problems with generalizability of model cost within a competition, as well as across Kaggle.
Not being able to fully recreate multiple submissions and get a better estimation of model cost is a serious limitation and this limits our ability to generalize model cost within a competition.
We were also only able to consider a single competition, meaning we cannot generalise to all of Kaggle.

Our estimations could be low.
CarbonTracker takes into account the energy grid running the code, and we ran ours in Denmark. We know teams came from all over the world, so their energy costs could well be higher.
Specifically speaking, the lower-bound could be low, since this value is based on a model that used transfer learning to train a single model, and on a smaller dataset, compared to the ensembles shown in the submitting teams.
The upper-bound estimation also could be low. This value was based on the cost of running the second place team's pipeline. But we were only able to evaluate 14 out of their 25 models, so the calculation is limited.

There are also reasons why our estimations could be high.
Using submission count as a proxy for training may be inaccurate. It could be that they "submitted" a notebook to debug it, but did not run training in between.

In addition, the time of day affects energy cost, and we did not limit our experiments to off-peak hours. We were basically forced to run our code when the queue for the HPC was free.
Given there was high interest in the subject at the time, and that a lot of people were unusually at home, and may have had time to learn new skills due to lockdowns across the world, there may have been more inexperienced competitors taking part, who would be submitting simpler solutions..



There were other, secondary findings

Code is not reproducible

Code competitions do not balance as intended

But the fun doesn’t end… there were other, secondary findings.

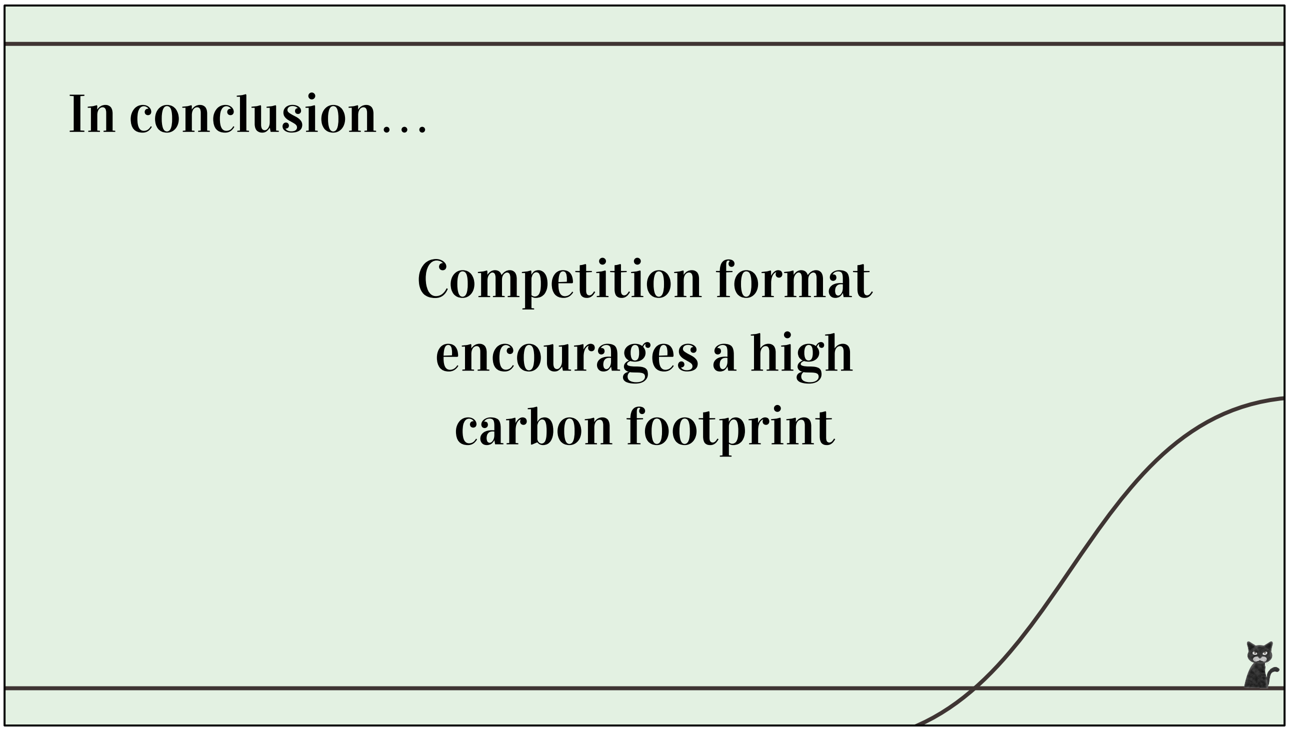
Solutions to the competition were not easily reproducible. Without being reproducible, this hinders the ability to make such solutions into a tool that can be used by doctors to solve the medical problems they claim to.
This seems to be a well-known problem with computer science research, with a number of authors offering solutions and guidelines.

We also found that code competitions on Kaggle are not as fair as they are intended to be.

The COVID-19 Detection competition has a run-time limit of less than or equal to 9 hours.
The team that we evaluated far exceeded that time limit by training externally and uploading weights.
When evaluating their code, it took more than 5 days running time, and this was for a single epoch (out of 5) at each step.
As others in the top 10 used a similar procedure, it shows participants are able to get around the restrictions imposed by the competition format, although not against the rules.

—---

NOTE: More than 5 days using sixteen CPU cores and an A100 40GB GPU
Over seventeen hours using sixteen CPU cores and two A100 40GB GPUs
This does not take into consideration the other two stages which we were unable to evaluate, or the additional time used in the submitted notebook itself.



In conclusion…

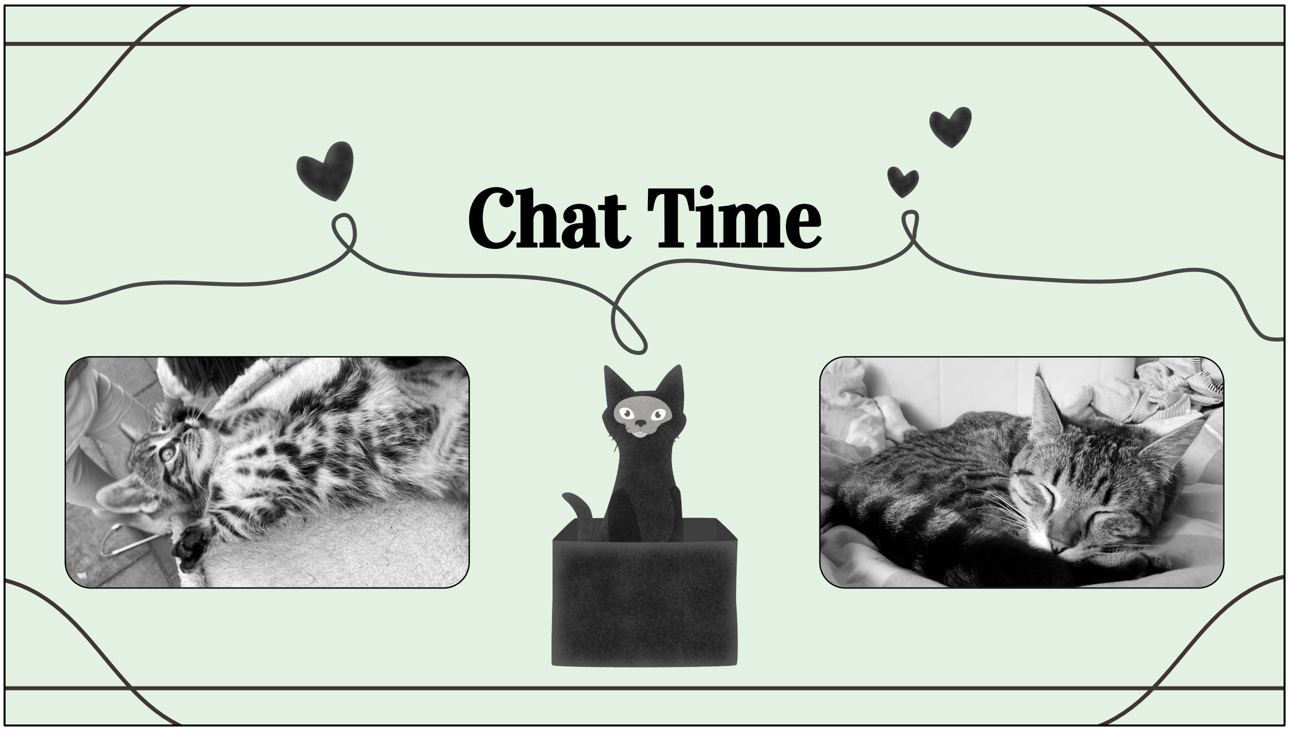
Competition format encourages a high carbon footprint

In conclusion
Competition format seems to encourage a high carbon footprint, with winning solutions not being significantly better than lower placed models.

But evaluation is difficult.

We would like Kaggle to encourage awareness of the carbon footprint of training models.

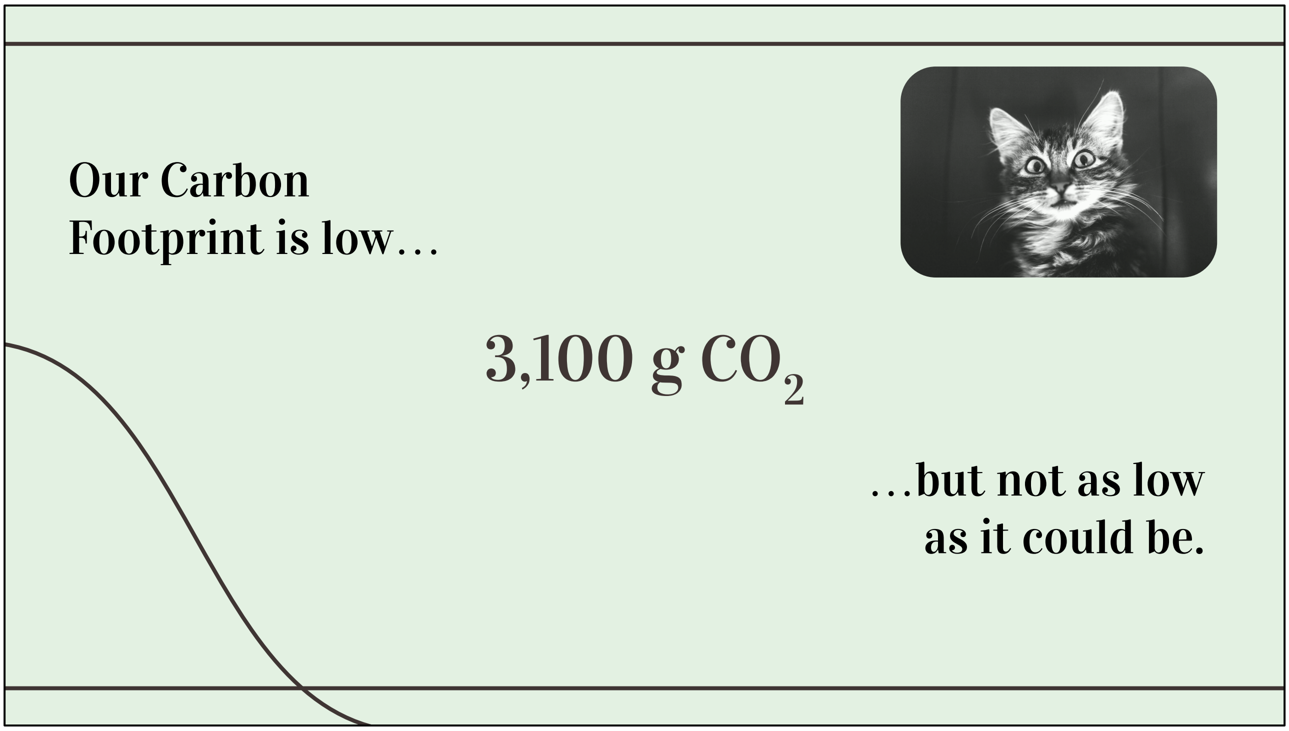
Participants could be required to use tools such as CarbonTracker, and report metrics that would improve the ability to measure a competitions carbon footprint, such as the number of times they trained a model, cumulative carbon cost, and final model carbon cost.
This change could encourage participants to be more carbon-conscious, and with this awareness it could decrease the carbon footprint of a competition, as well as make the evaluation of it easier.



Chat Time

Thank you for listening,
Did you have any questions?

Additional slides



Our Carbon Footprint is low…

3,100 g CO_2

…but not as low as it could be.

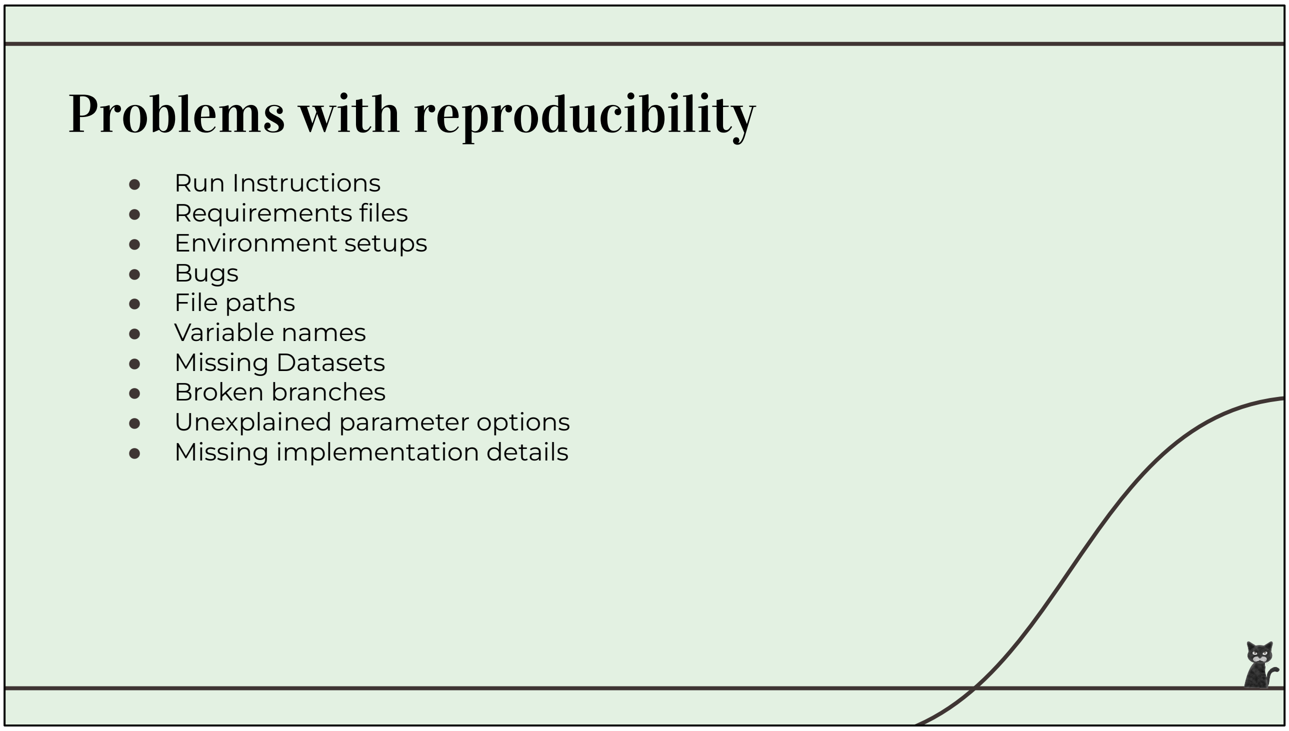
To calculate our carbon footprint:

- number of times we trained each model
- how much of the training had completed.

For example, we ran the training of the ResNet50 model on the Thyroid dataset five times but only once on the Knee dataset during development and testing. Therefore we multiply the cost of the first model by five, and the cost of the latter by one when calculating our total cost.

Our carbon footprint is estimated to be 3100 g CO_2 , or 29 km driven by car.

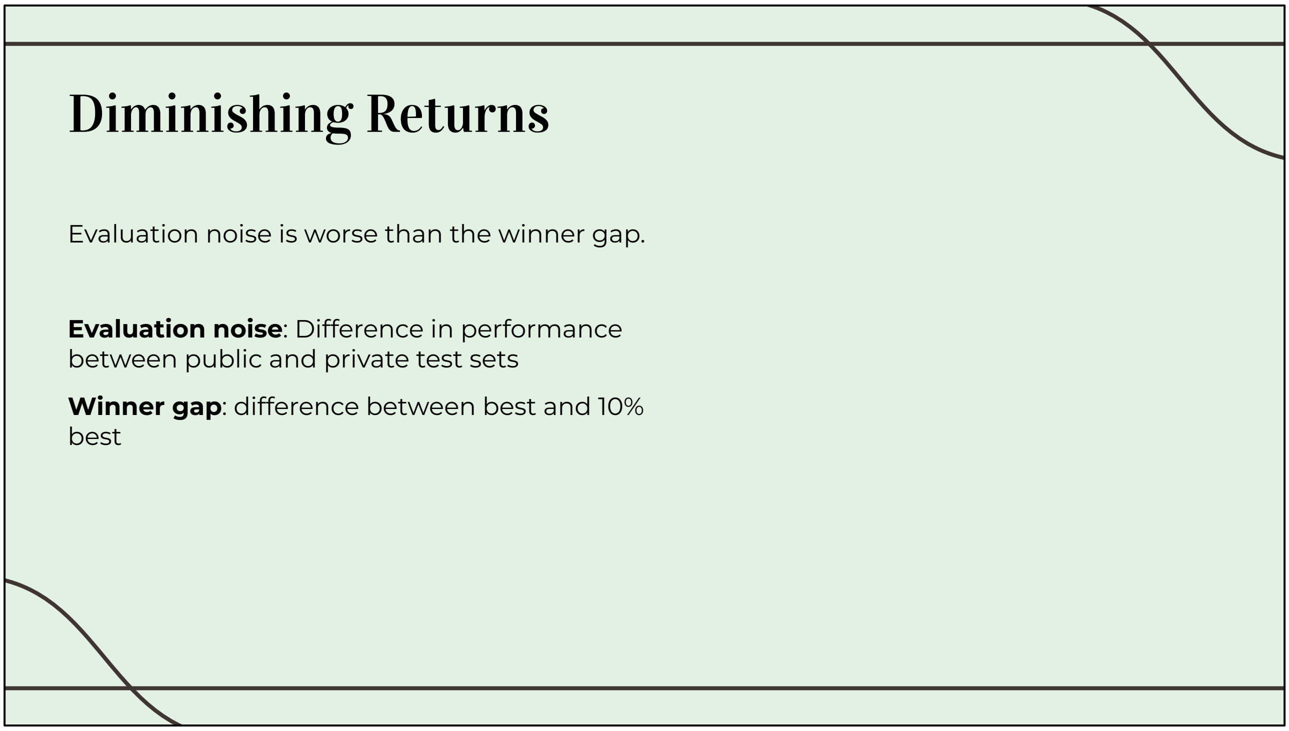
If everything ran smoothly without needing to fix any problems along the way, our CO_2 emissions would be 1483 g, the equivalent to 14 km driven by a car.
The result of our carbon footprint is insufficient. There is more that goes into evaluating the carbon footprint of ML than tracking the training of a model. In our study, we used a high performance cluster (HPC) for data storage and executing code, and used personal laptops for development and testing. The costs of activities outside of training a model were not tracked, and subsequently not added into our total carbon footprint.



Problems with reproducibility

- Run Instructions
- Requirements files
- Environment setups
- Bugs
- File paths
- Variable names
- Missing Datasets
- Broken branches
- Unexplained parameter options
- Missing implementation details

Problems we ran into are:
• Lack of comprehensible instructions.
• Lack of proper requirements files.
• Variation in local environment setups.
• Typing errors and other mistakes present in code.
• File paths that need to be modified are not easy to find for customization.
• Discrepancies in variable names throughout the source code.
• Datasets no longer available.
• Necessary branches in a git repository being broken.
• Unexplained parameter options.
• Missing implementation details.



Diminishing Returns

Evaluation noise is worse than the winner gap.

Evaluation noise: Difference in performance between public and private test sets

Winner gap: difference between best and 10% best

Too strong a focus on benchmark performance can lead to **diminishing returns**, where increasingly large efforts achieve smaller and smaller performance gains.
- when the evaluation noise is worse than the winner gap. In other words, the gains made by the top 10% of methods are smaller than the expected noise when evaluating a method.

Evaluation noise –the spread of performance differences between the public and private test sets.
Winner gap—the difference in performance between the best algorithm, and the "top 10%" algorithm.

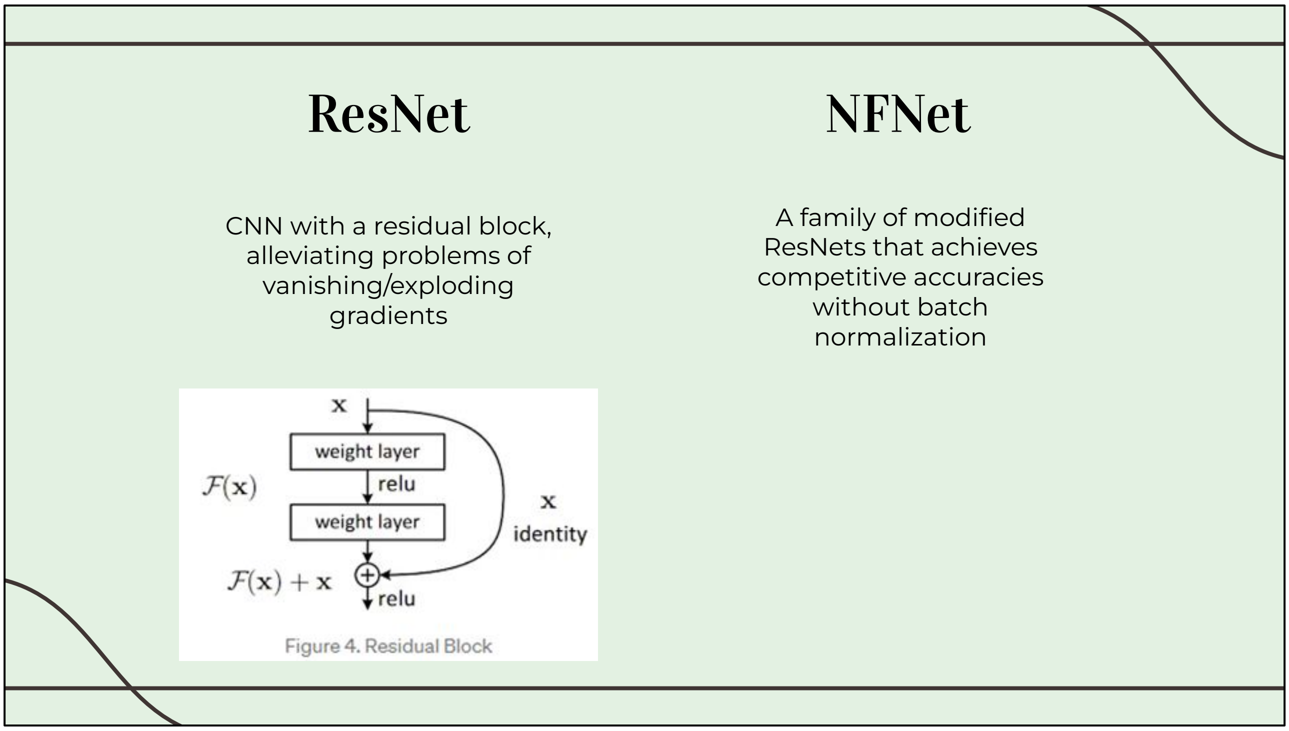
NOT diminishing returns: a winner gap larger than the evaluation noise, meaning that the winning method made substantial improvements compared to the 10% competitor.

When the winner gap is larger than the evaluation noise - the winning method made substantial improvements compared to the 10% competitor.

—---

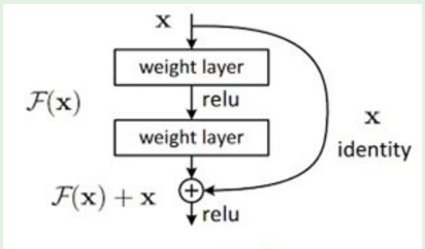
Here the nerve **segmentation** challenge is the only one shown that did not have diminishing returns. In other words,the winner made significant improvements. This can be seen since the vertical brown bar is beyond the expected evaluation noise.

Whereas the **lung** cancer and **schizophrenia** have diminishing returns, with the winner (vertical brown bar) well within the evaluation noise (blue violin)
For the **pneumothorax** challenge, we can see that performance on the private set (vertical blue bar) is worse than on the public set (vertical gray bar), revealing an overfit larger than the winner gap.



ResNet

CNN with a residual block, alleviating problems of vanishing/exploding gradients


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
Figure 4. Residual Block

NFNet

A family of modified ResNets that achieves competitive accuracies without batch normalization

The traditional CNN contains a **convolutional layer**, an **activation function,** and a **max pooling** operation. Due to the vanishing/exploding gradient problems, it has been shown that increasing the depth of a network does not necessarily improve its classification performance.

These problems have been alleviated with the introduction of the **ResNet** architectures.
These are characterized by a new NN layer, the so called Residual Block with the 'Skip Connection' identity mapping.
The identity mapping does not have any parameters and is just there to add the output from the previous layer to the layer ahead.
In ResNet-like architectures, batch normalization is often applied in the residual branch, which has the effect of reducing the scale of activations on the residual branch (compared to the skip branch).
This stabilizes the gradient early in training, enabling the training of significantly deeper networks.

However, there are downsides to batch normalization; one being it is expensive.
So then we have **NFNets**; a family of modified ResNets that achieves competitive accuracies without batch normalization.

To do so, it applies 3 different techniques:

- Modified residual branches and convolutions with Scaled Weight Standardization

- Adaptive Gradient Clipping
- Architecture optimization for improved accuracy and training speed

Modified residual branches: Suppress the scale of the activations on the residual branch by using 2 scalers (α and β) to scale the activations at the start and end of the residual branch.
Adaptive Gradient clipping: clip the gradient when the gradient-to-weight ratio exceeds a certain threshold.

People also used **EfficientNet**.
EfficientNet is a CNN architecture and scaling method that uniformly scales all dimensions of depth/width/resolution using a set of fixed scaling coefficients. For example, if we want to use 2^N times more computational resources, then we can simply increase the network depth by a^N, width by
B^N, and image size by Y^N, where are constant coefficients determined by a small grid search on the original small model.

Also **Yolo**
YOLO (You Only Look Once) is a popular object detection model known for its speed and accuracy.

References

https://medium.com/@enrico.randellini/image-classification-resnet-vs-efficientnet-vs-efficientnet-v2-vs-compact-convolutional-c205838bbf49

https://towardsdatascience.com/nfnets-explained-deepminds-new-state-of-the-art-image-classifier-10430c8599ee