Predicting the performance of systems through Deep Learning

Stephen McGough

stephen.mcgough@newcastle.ac.uk

Huawei Global Software Technology Summit

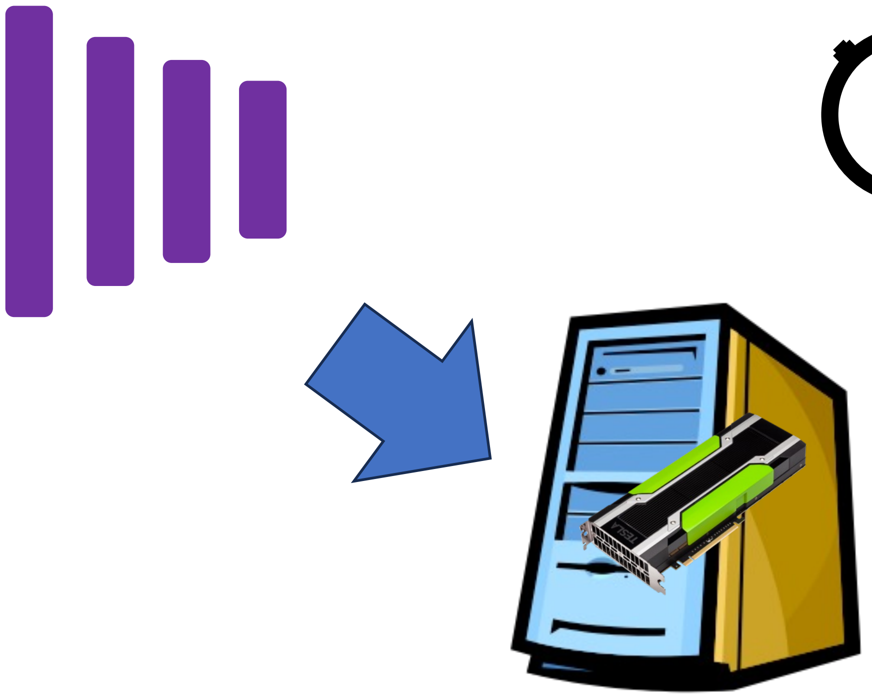
1st June 2023

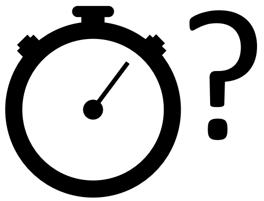
Edinburgh, UK

Outline

- Predicting Deep Learning execution time
  - Problem and Approach
  - Predicting Execution time for individual operations
  - Predicting for a full model
- Predicting SPEC 2017 performance
  - Problem and Approach
  - The Data
  - Data preparation
  - Deep Learning models
  - Results
- Conclusions

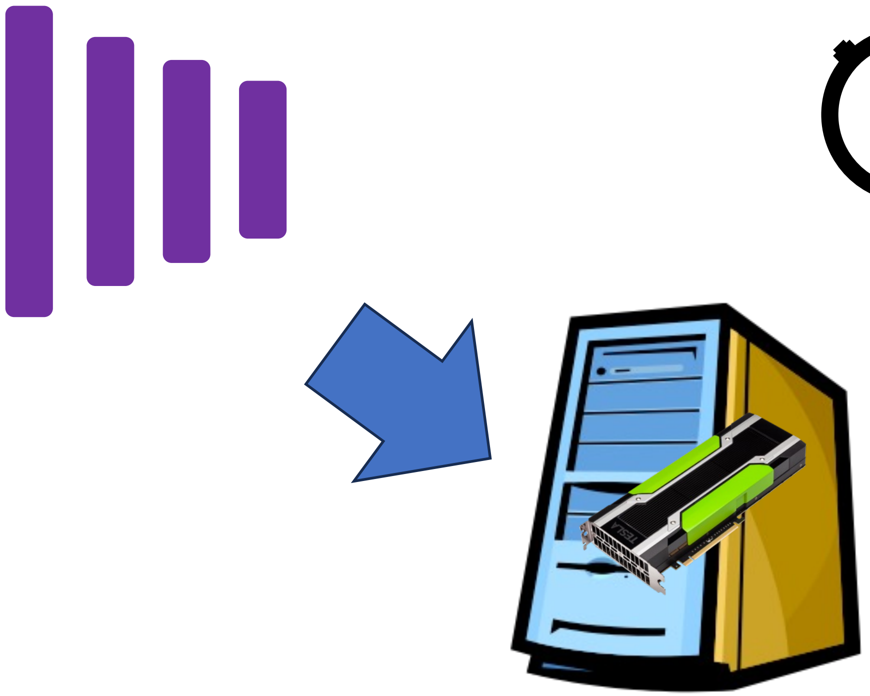
Problem in Predicting

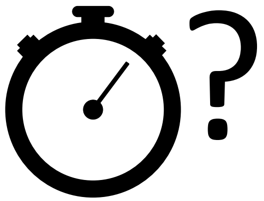




- Inference time
- Epoch training time

Problem in Predicting

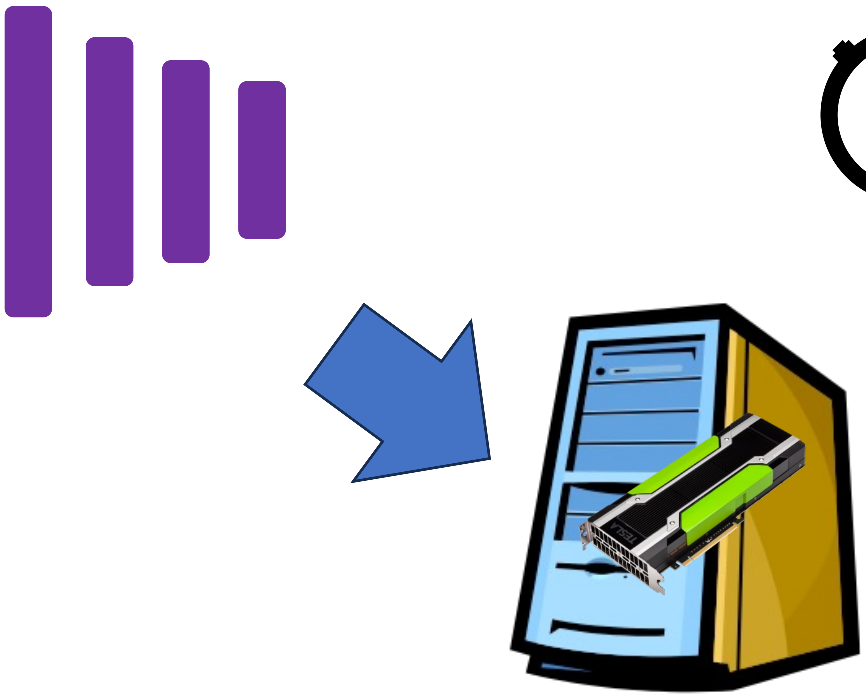


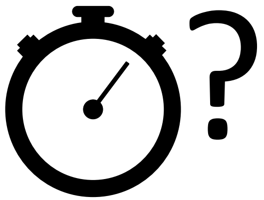


- Inference time
- Epoch training time

- Just Linear operations right?
- Matrix x Matrix = $O(n^3)$
- Matrix x Vector = $O(n^2)$

Problem in Predicting



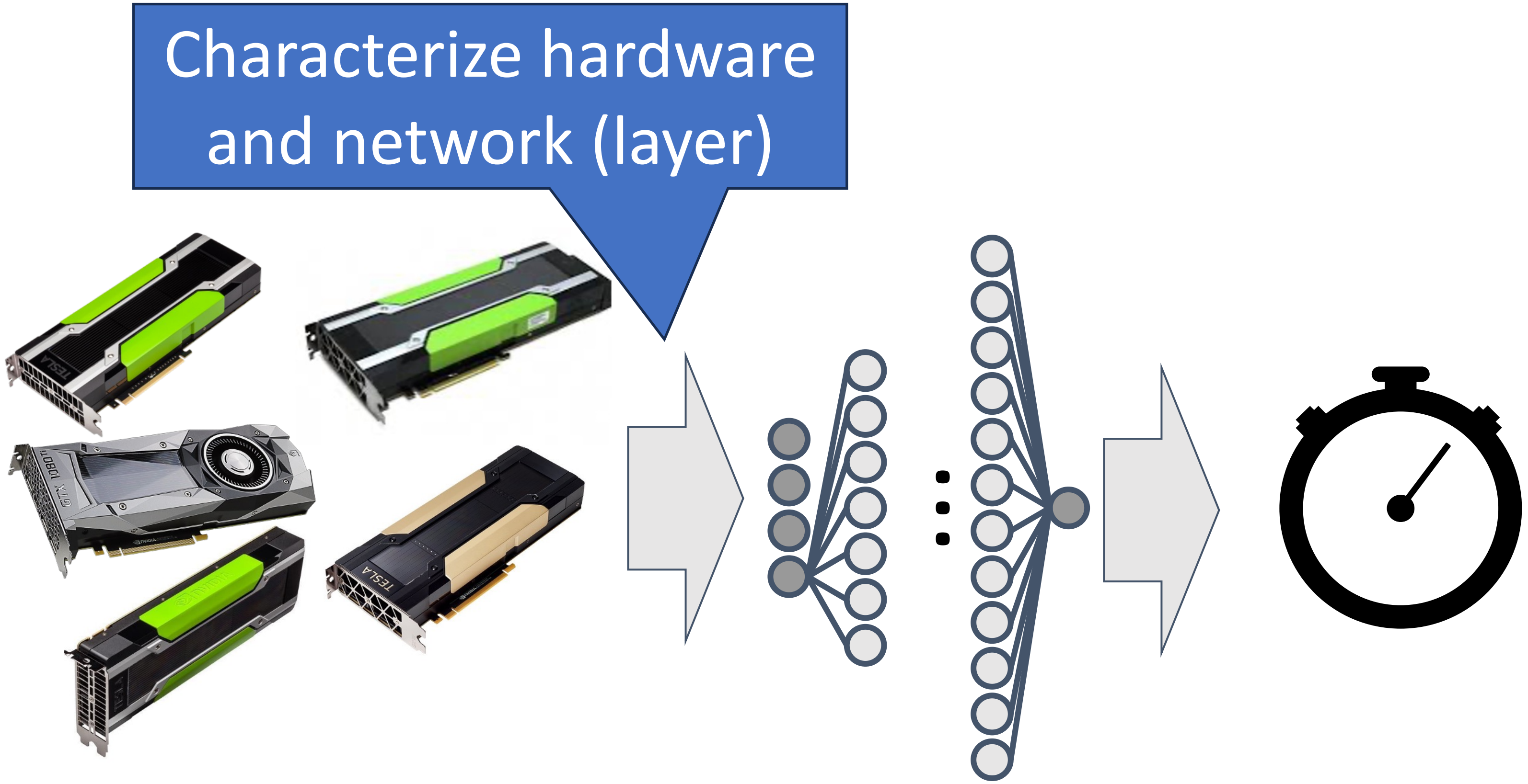


- Inference time
- Epoch training time

- ~~Just linear operations right?~~
- ~~Matrix x Matrix = $O(n^3)$~~
- ~~Matrix x Vector = $O(n^2)$~~

- Ignores non-linearity of hardware
- Issues between hardware versions

Approach: Benchmark and predict



Why Deep Learning?

Analytical approaches based on the number of FLOPs and data size often neglect nonlinearities from model or hardware features

Data transfer

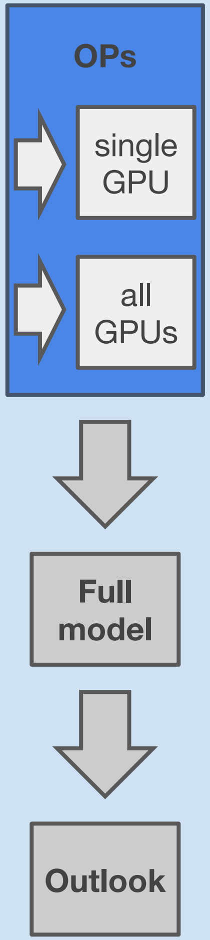
backpropagation with different optimisers

suboptimal hardware utilisation (especially of GPUs)

Factors influencing the execution time

Model features

- Required number floating/fixed point operations
- temporal sequence of operations (serial/parallel)
- Amount of data
- Optimiser



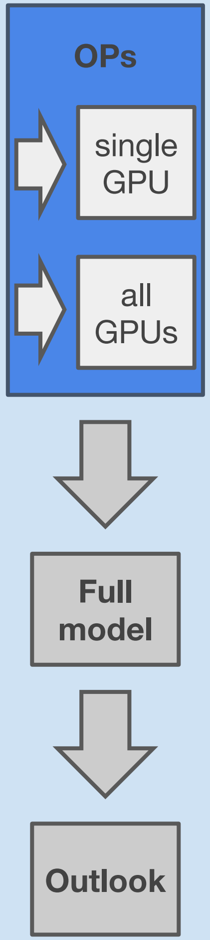
Factors influencing the execution time

Model features

- Required number floating/fixed point operations
- temporal sequence of operations (serial/parallel)
- Amount of data
- Optimiser

Hardware features

- Clock frequency
- Number of processing units / shader units
- Available memory bandwidth



Factors influencing the execution time

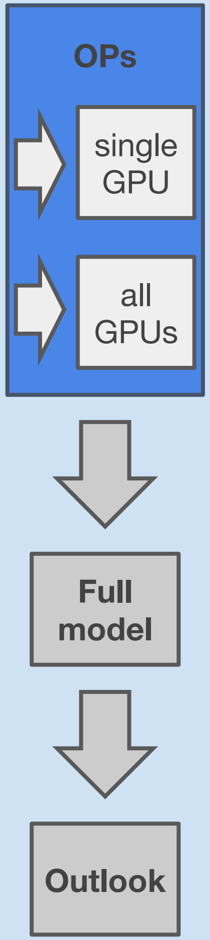
Model features

- Required number floating/fixed point operations
- temporal sequence of operations (serial/parallel)
- Amount of data
- Optimiser

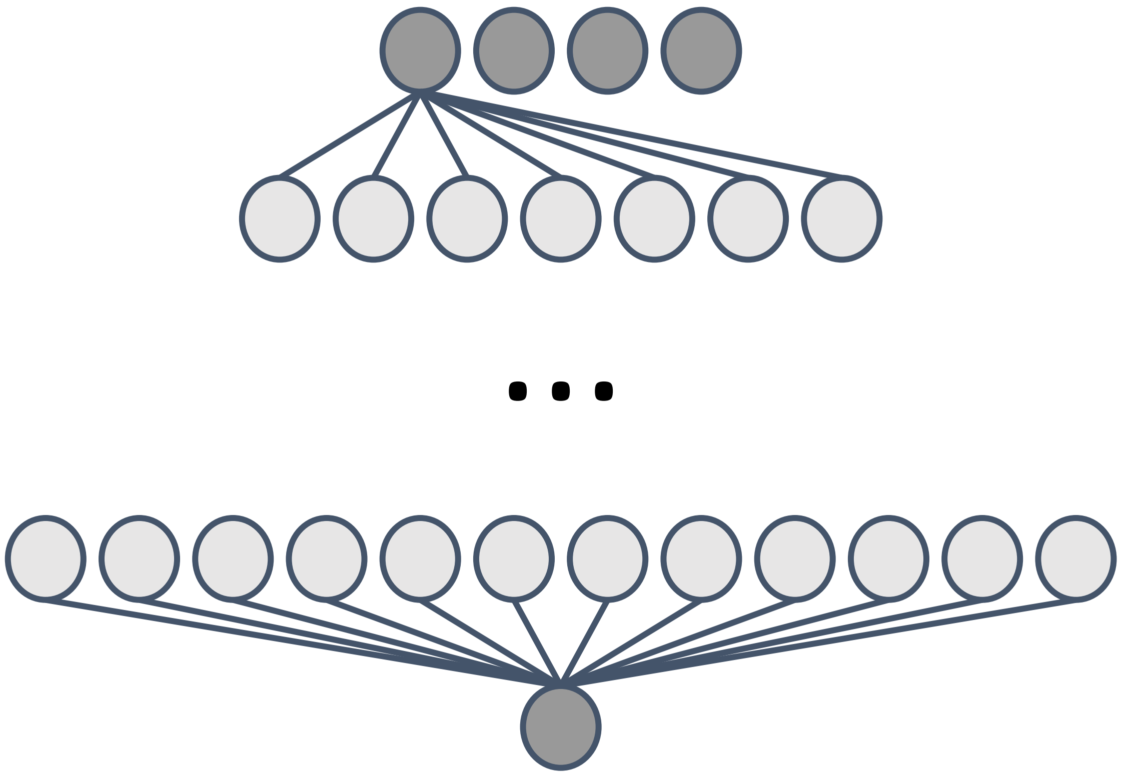
Hardware features

- Clock frequency
- Number of processing units / shader units
- Available memory bandwidth

- Utilisation of the available processing units and memory bandwidth



An ML model to predict execution time

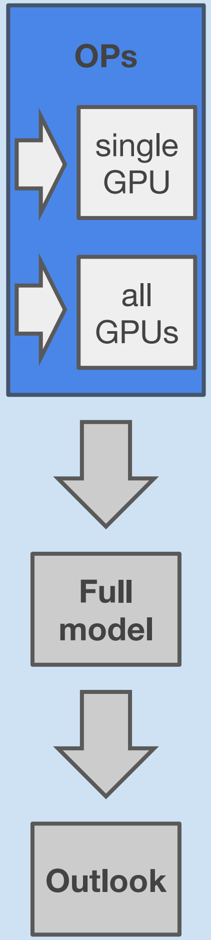


Model features: matrix size, input dimensions, optimiser, ...

Hardware features: clock frequency, number of cores, memory bandwidth, ...

Hidden layers

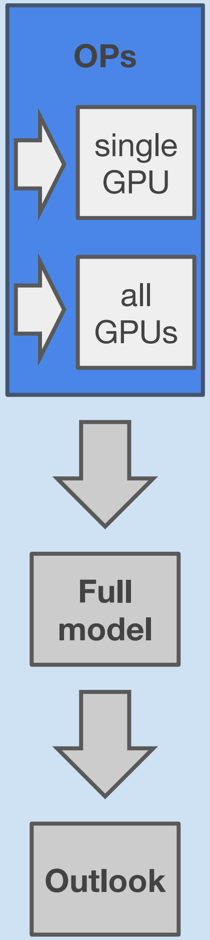
Execution time prediction



Data generation

Benchmark fundamental operations on different GPUs

- Convolution and fully connected layers (vector-matrix multiplication)
- Random parameters
- 5 Iterations → Use mean time
- 80% - 10% - 10% split for train / test / validation



Outline

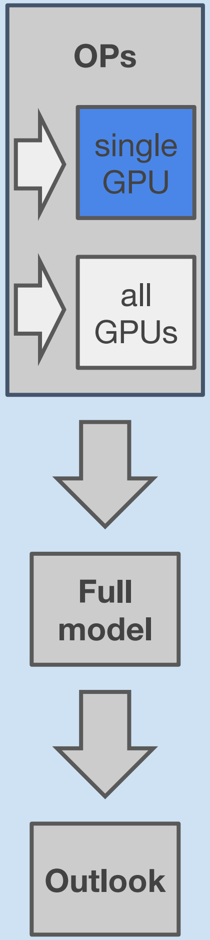
- Predicting Deep Learning execution time
  - Problem and Approach
  - Predicting Execution time for individual operations
  - Predicting for a full model
- Predicting SPEC 2017 performance
  - Problem and Approach
  - The Data
  - Data preparation
  - Deep Learning models
  - Results
- Conclusions

Single GPU models for convolutions

A model trained on data from a single GPU

Input features

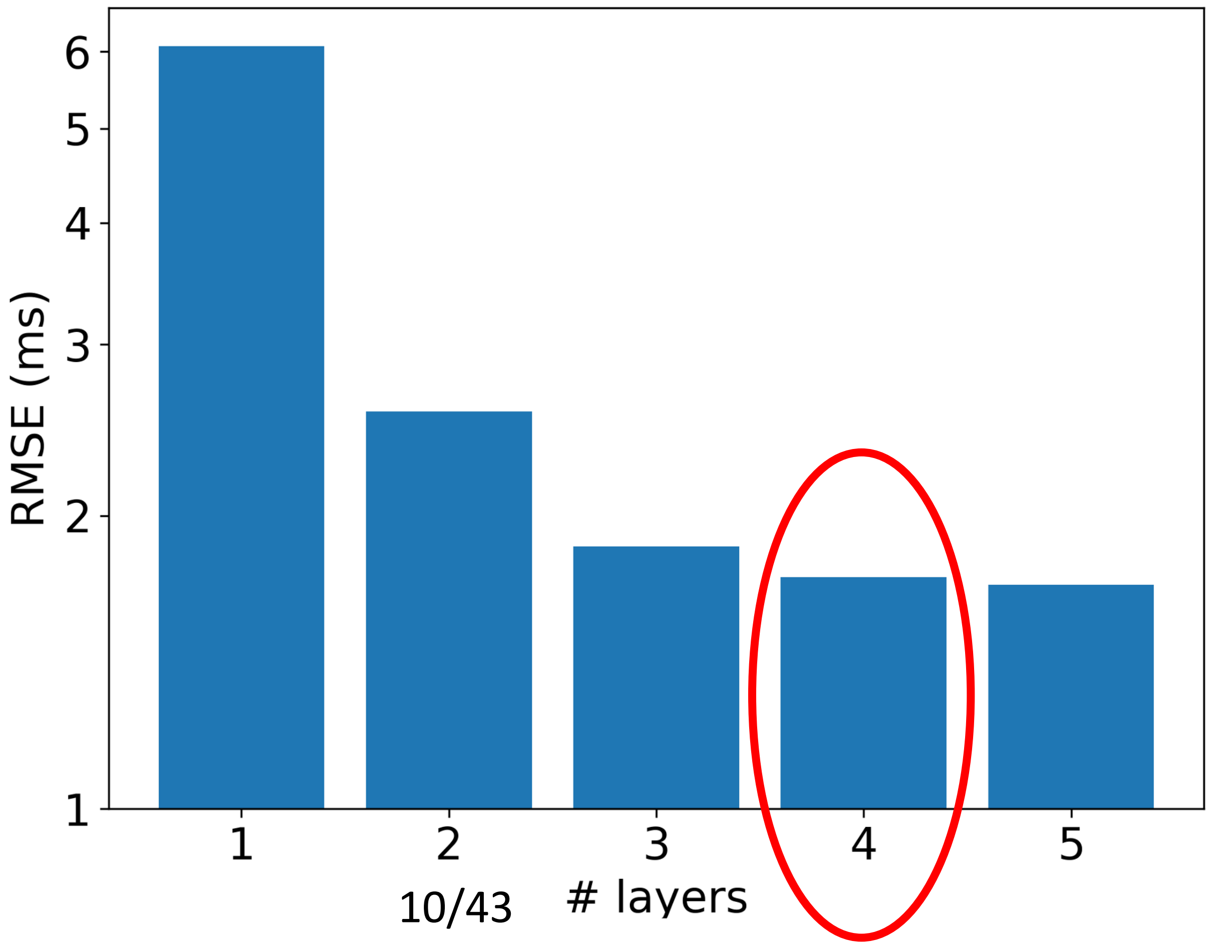
- Batch size
- Matrix size
- Kernel size
- Input dimension
- Output dimension
- Input padding
- Kernel strides
- Optimiser (one-hot encoded)
- Activation function (one-hot encoded)

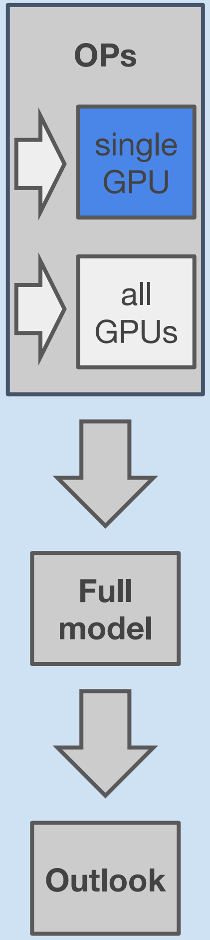


Predictive Model Architecture

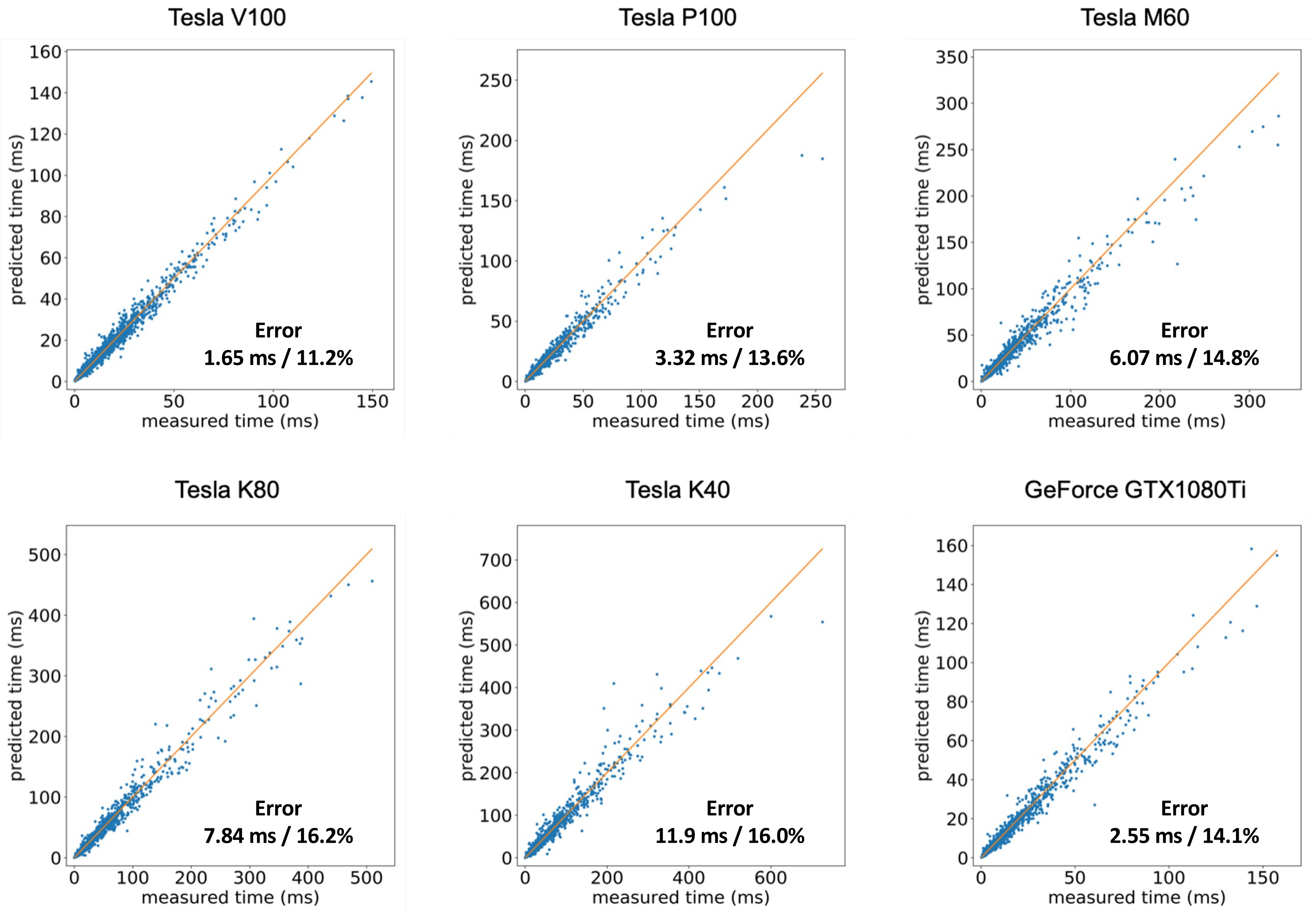
Design

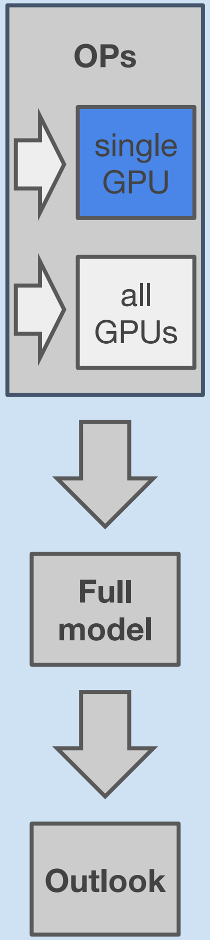
- 4 Layer MLP
- Dropout Layer
- Adam Optimiser





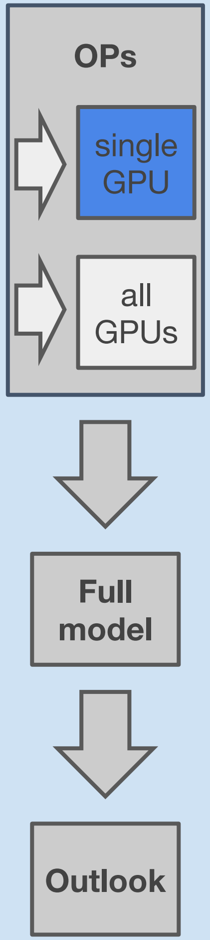
Results


Tesla V100
predicted time (ms)
measured time (ms)
Error
1.65 ms / 11.2%
Tesla P100
predicted time (ms)
measured time (ms)
Error
3.32 ms / 13.6%
Tesla M60
predicted time (ms)
measured time (ms)
Error
6.07 ms / 14.8%
Tesla K80
predicted time (ms)
measured time (ms)
Error
7.84 ms / 16.2%
Tesla K40
predicted time (ms)
measured time (ms)
Error
11.9 ms / 16.0%
GeForce GTX1080Ti
predicted time (ms)
measured time (ms)
Error
2.55 ms / 14.1%


OPs
single GPU
all GPUs
Full model
Outlook

Comparison with Linear model

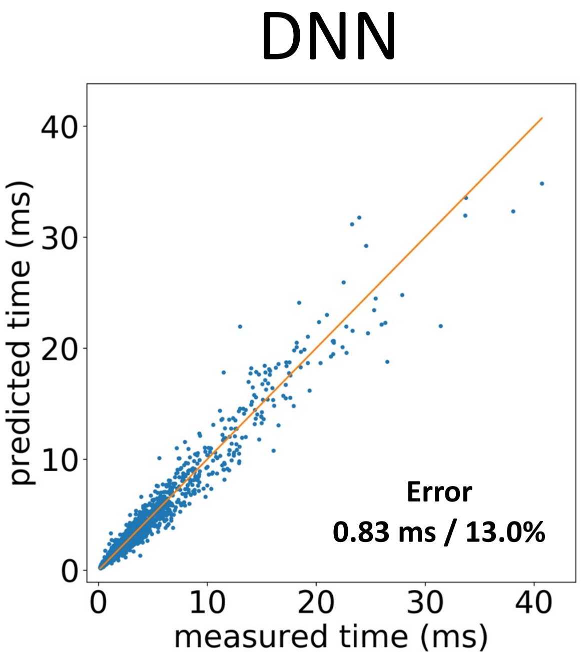
- Separate cases for forward and forward+backward pass with a specific optimiser
- Number of required floating point operations as additional feature for linear model

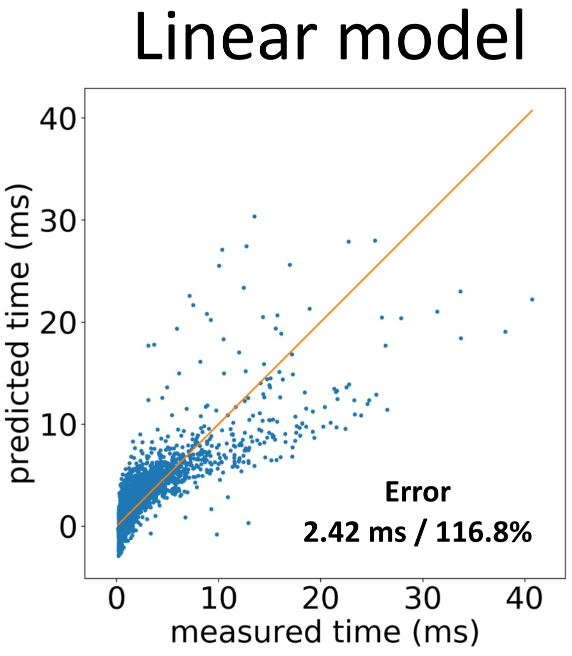


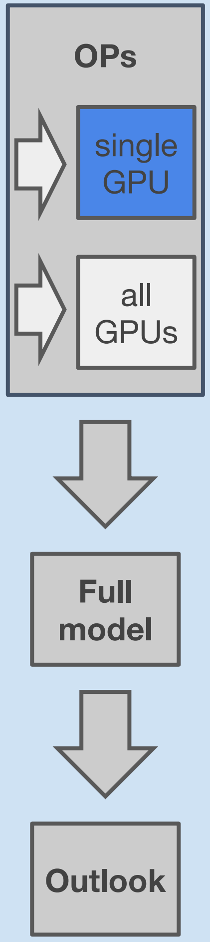
Comparison with Linear model

- Separate cases for forward and forward+backward pass with a specific optimiser
- Number of required floating point operations as additional feature for linear model

•Forward pass only



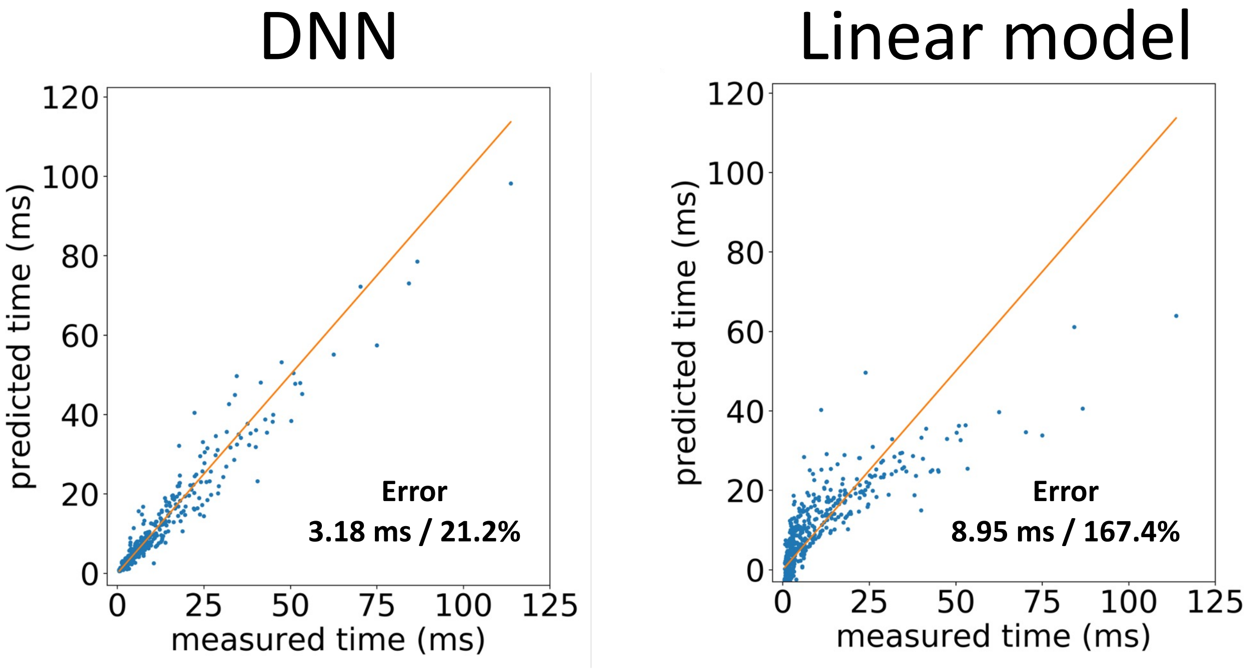


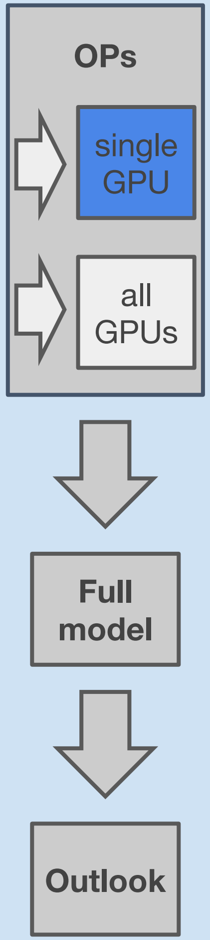


Comparison with Linear model

- Separate cases for forward and forward+backward pass with a specific optimiser
- Number of required floating point operations as additional feature for linear model

- **Forward and backward pass with SGD**





Outline

- Predicting Deep Learning execution time
  - Problem and Approach
  - Predicting Execution time for individual operations
  - Predicting for a full model
- Predicting SPEC 2017 performance
  - Problem and Approach
  - The Data
  - Data preparation
  - Deep Learning models
  - Results
- Conclusions

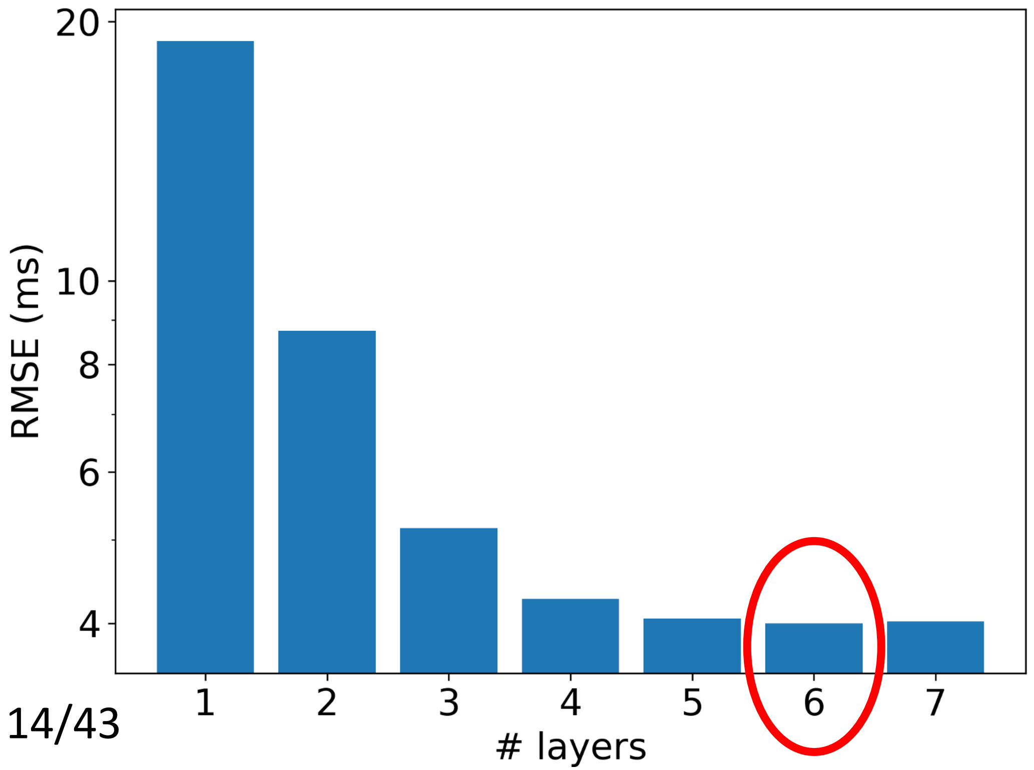
A General Model

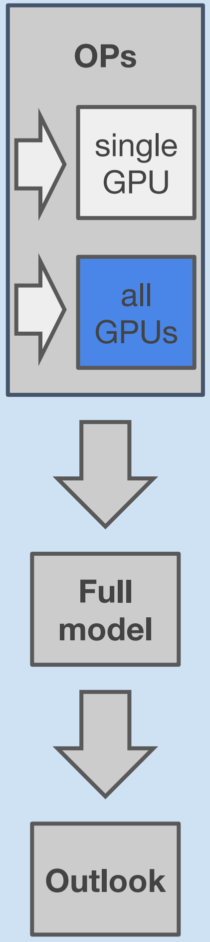
- **A model trained on data from multiple GPUs**

Additional input features

- GPU clock frequency (MHz)
- number of GPU shader units
- GPU memory bandwidth (GB/s)

•Model architecture?

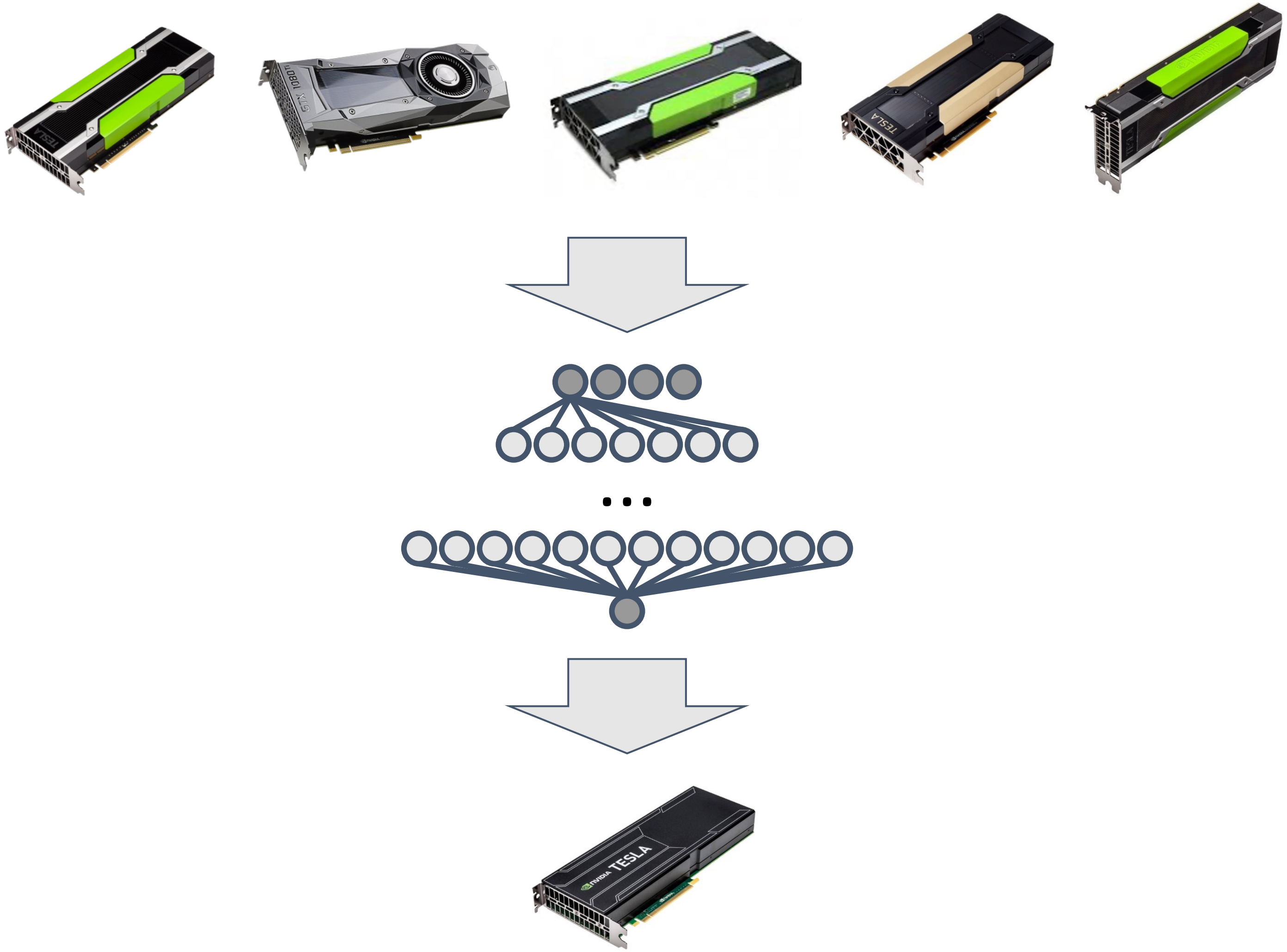


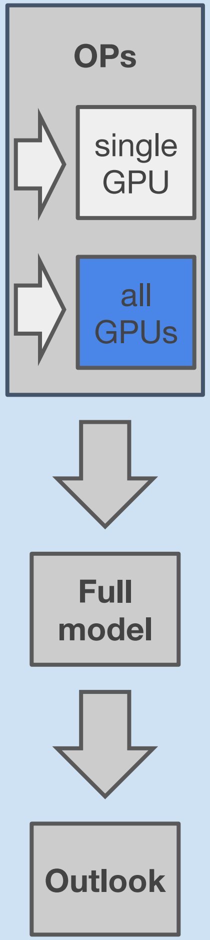


Can we predict performance of unseen hardware?

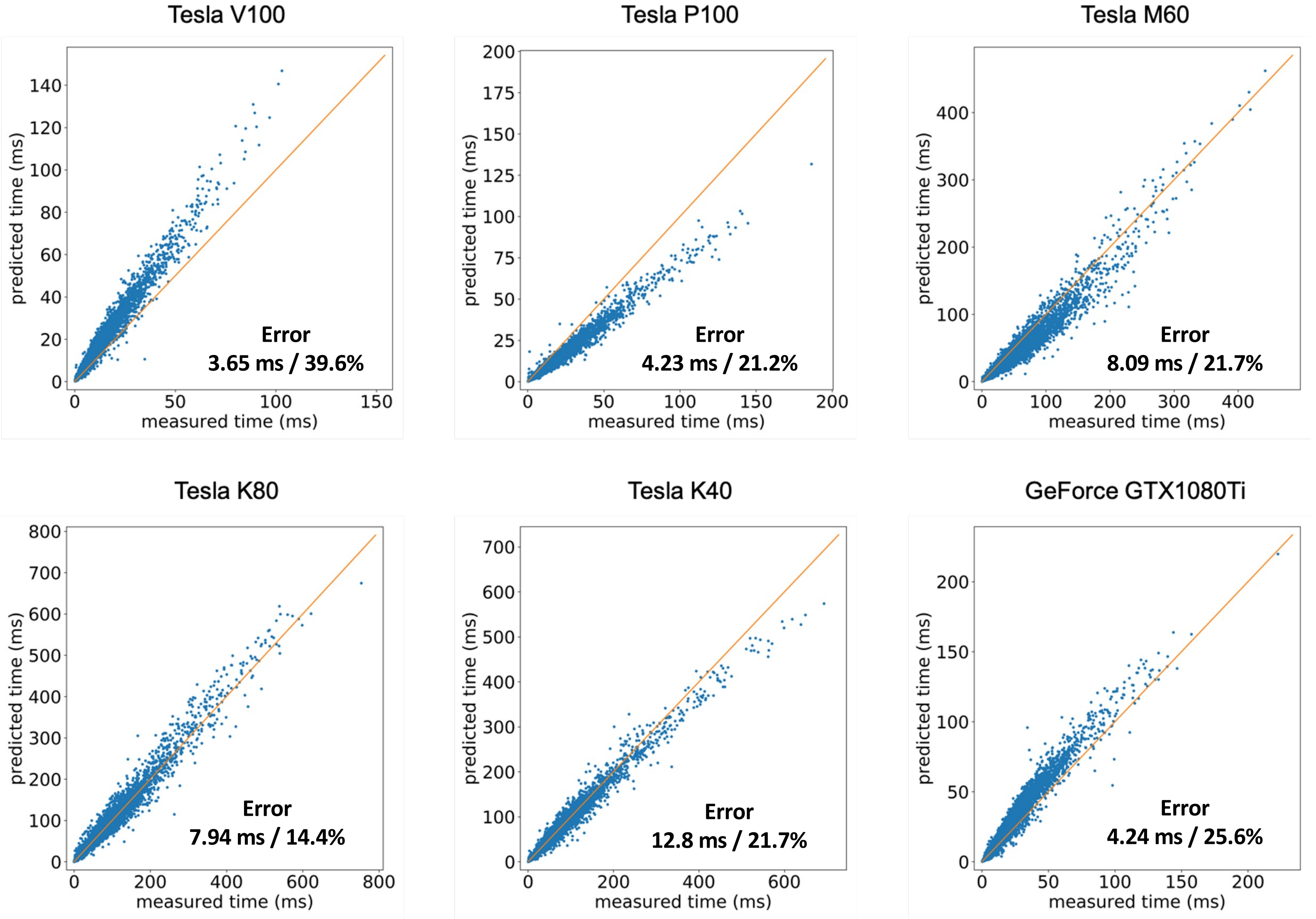
Cross-validation

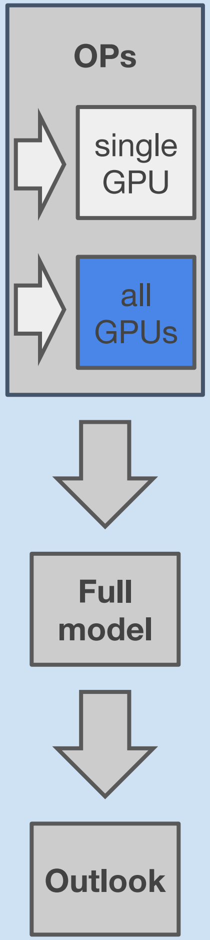
- Train a model on data from five GPUs
- Predict the execution time for the sixth GPU
- Repeat for every combination





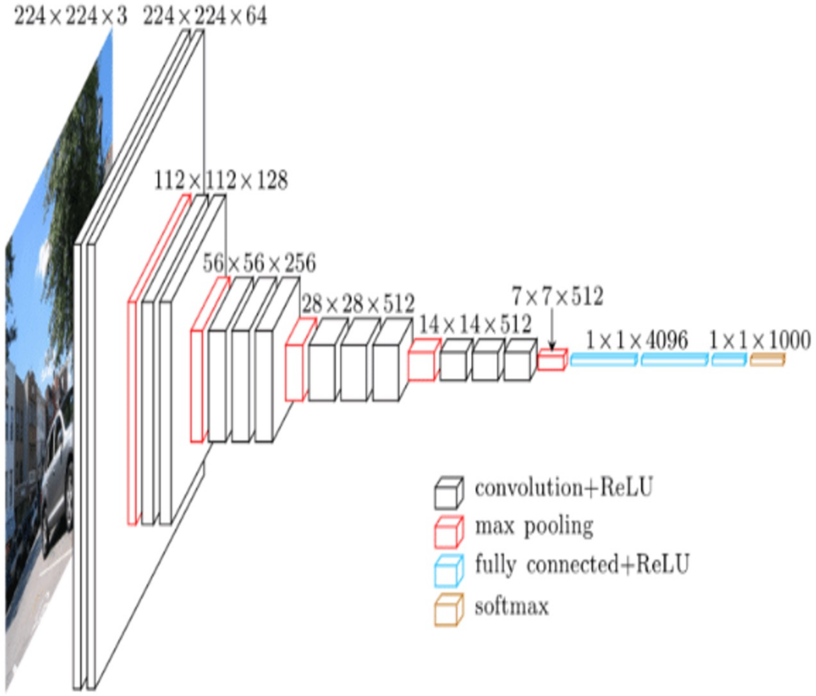
Can we predict performance of unseen hardware?



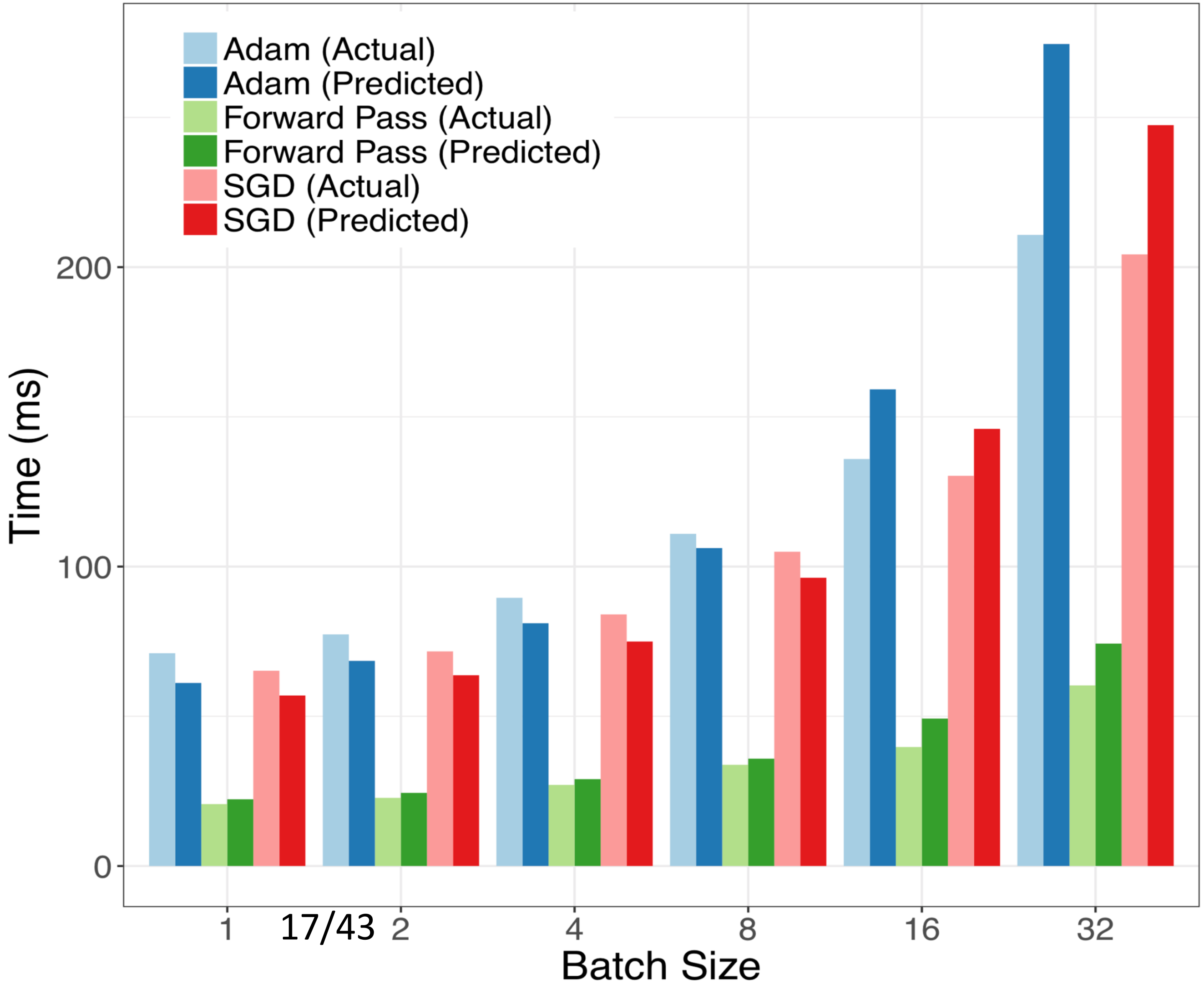


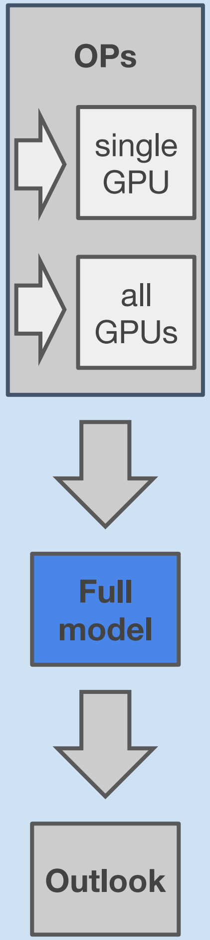
Test for full models

VGG-16 for image classification



Simonyan & Andrew Zisserman: Very Deep Convolutional Networks for Large-Scale Image Recognition





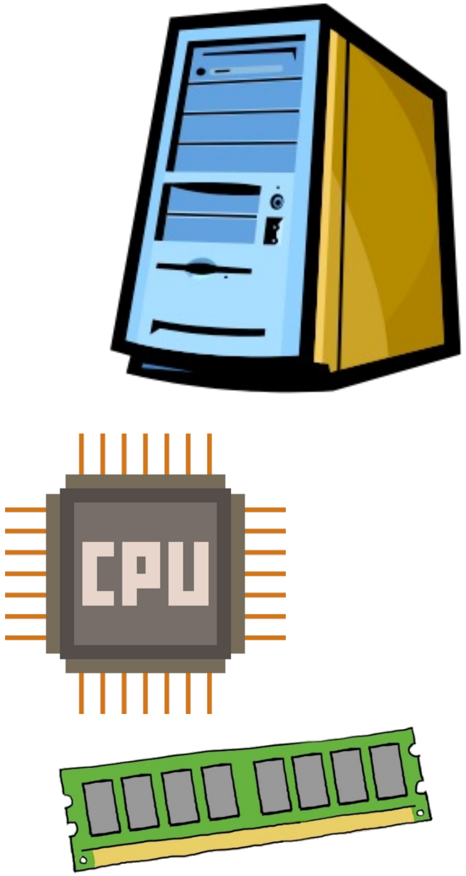
Outline

- Predicting Deep Learning execution time
  - Problem and Approach
  - Predicting Execution time for individual operations
  - Predicting for a full model
- Predicting SPEC 2017 performance
  - Problem and Approach
  - The Data
  - Data preparation
  - Deep Learning models
  - Results
- Conclusions

Predicting SPEC CPU 2017 scores for new computers

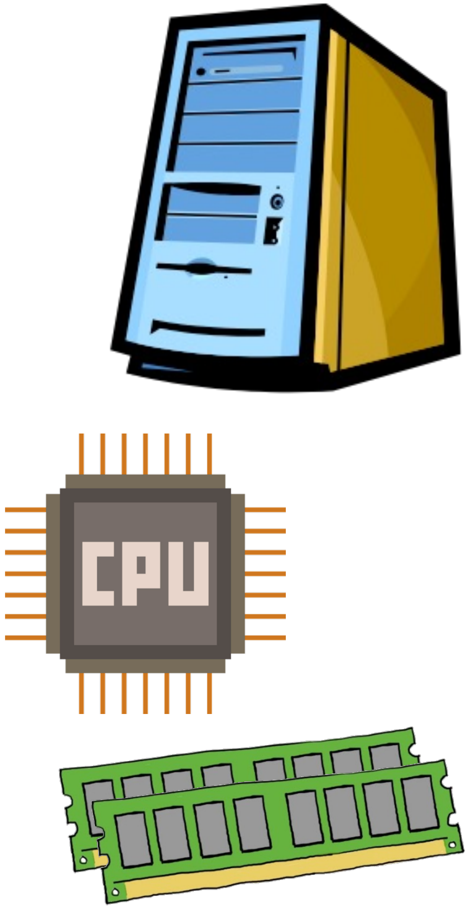
SPEC CPU 2017 score

Predicting SPEC CPU 2017 scores for new computers



SPEC CPU 2017 score

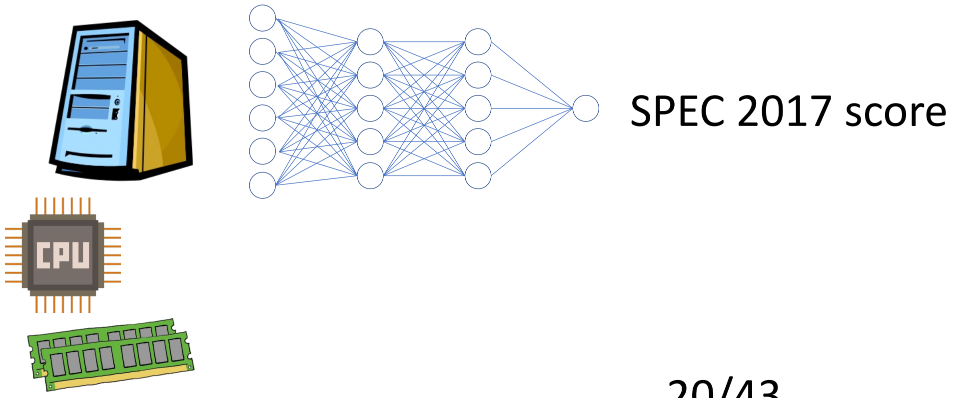
Predicting SPEC CPU 2017 scores for new computers



SPEC CPU 2017 score

Filling in the gaps with Machine Learning

- Benchmarking systems is costly
  - Time to conduct tests
  - Financial (hardware + software)
- Machine Learning is promising alternative to building and testing
  - Especially Deep Learning
- We demonstrate the potential of deep learning for predicting performance
  - using Multi-layer Perceptrons and Convolutional Neural Networks



Outline

- Predicting Deep Learning execution time
  - Problem and Approach
  - Predicting Execution time for individual operations
  - Predicting for a full model
- Predicting SPEC 2017 performance
  - Problem and Approach
  - The Data
  - Data preparation
  - Deep Learning models
  - Results
- Conclusions

Features in SPEC CPU 2017 dataset

Data Type	Column
String	Benchmark, Hardware Vendor, System, Processor, CPU(s) Orderable, 1st Level Cache, 2nd Level Cache, 3rd Level Cache, Other Cache, Storage, Operating System, File System, Compiler, License, Tested By, Test Sponsor
Numerical	Peak Result, Base Result, Energy Peak Result, Energy Base Result, # Cores, # Chips, Memory, # Enabled Threads Per Core, Processor MHz
Binary	Parallel
Ternary	Base Pointer Size
Quaternary	Peak Pointer Size
Date (mon-yyyy)	HW Avail, SW Avail, Test Date, Published, Updated
Text	Disclosures

34 attributes / features

Features in SPEC CPU 2017 dataset

Data Type	Column
String	Benchmark, Hardware Vendor, System, Processor, CPU(s) Orderable, 1st Level Cache, 2nd Level Cache, 3rd Level Cache, Other Cache, Storage, Operating System, File System, Compiler, License, Tested By, Test Sponsor
Numerical	Peak Result, Base Result, Energy Peak Result, Energy Base Result, # Cores, # Chips, Memory, # Enabled Threads Per Core, Processor MHz
Binary	Parallel
Ternary	Base Pointer Size
Quaternary	Peak Pointer Size
Date (mon-yyyy)	HW Avail, SW Avail, Test Date, Published, Updated
Text	Disclosures

34 attributes / features

Best result with optimization

Features in SPEC CPU 2017 dataset

Data Type	Column
String	Benchmark, Hardware Vendor, System, Processor, CPU(s) Orderable, 1st Level Cache, 2nd Level Cache, 3rd Level Cache, Other Cache, Storage, Operating System, File System, Compiler, License, Tested By, Test Sponsor
Numerical	Peak Result, Base Result, Energy Peak Result, Energy Base Result, # Cores, # Chips, Memory, # Enabled Threads Per Core, Processor MHz
Binary	Parallel
Ternary	Base Pointer Size
Quaternary	Peak Pointer Size
Date (mon-yyyy)	HW Avail, SW Avail, Test Date, Published, Updated
Text	Disclosures

34 attributes / features

Best result with optimization

Result with no optimization

Features in SPEC CPU 2017 dataset

Data Type	Column
String	Benchmark, Hardware Vendor, System, Processor, CPU(s) Orderable, 1st Level Cache, 2nd Level Cache, 3rd Level Cache, Other Cache, Storage, Operating System, File System, Compiler, License, Tested By, Test Sponsor
Numerical	Peak Result, Base Result, Energy Peak Result, Energy Base Result, # Cores, # Chips, Memory, # Enabled Threads Per Core, Processor MHz
Binary	Parallel
Ternary	Base Pointer Size
Quaternary	Peak Pointer Size
Date (mon-yyyy)	HW Avail, SW Avail, Test Date, Published, Updated
Text	Disclosures

34 attributes / features

Best result with optimization

Result with no optimization

What we'll predict

SPEC CPU 2017 Data example

Benchmark = 'CINT2017',

Hardware Vendor = 'ASUSTeK Computer Inc.',

System = 'ASUS ESC4000A-E10(KRPG-U8) Server System 2.60 GHz, AMD EPYC 7H12',

Peak Result = 9.09,

Base Result = 8.87,

Energy Peak Result = 0.0,

Energy Base Result = 0.0,

Cores = 64,

Chips = 1,

Enabled Threads Per Core = 2,

Processor = 'AMD EPYC 7H12'

Processor MHz = 2600

CPU(s) Orderable = '1 chip',

Parallel = 'Yes',

Base Pointer Size = '64-bit',

Peak Pointer Size = '32/64-bit',

1st Level Cache = '32 KB I + 32 KB D on chip per core',

2nd Level Cache = '512 KB I+D on chip per core',

3rd Level Cache = '256 MB I+D on chip per chip, 16 MB shared / 4 cores',

Other Cache = 'None',

Memory = '512 GB (8 x 64 GB 2Rx4 PC4-3200AA-R)',

Storage = '1 x 480 GB SATA SSD',

Operating System = 'Ubuntu 19.04 (x86_64), Kernel 5.0.0-20-generic',

File System = 'ext4',

Compiler = 'C/C++/Fortran: Version 2.0.0 of AOCC',

HW Avail = 'Jul-2020',

SW Avail = 'Jun-2019',

License = 9016,

Tested By = 'ASUSTeK Computer Inc.',

Test Sponsor = 'ASUSTeK Computer Inc.',

Test Date = 'Jun-2020',

Published = 'Jul-2020',

Updated = 'Jul-2020',

Disclosures = 'HTML CSV PDF PS Text Config'

Outline

- Predicting Deep Learning execution time
  - Problem and Approach
  - Predicting Execution time for individual operations
  - Predicting for a full model
- Predicting SPEC 2017 performance
  - Problem and Approach
  - The Data
  - Data preparation
  - Deep Learning models
  - Results
- Conclusions

Cleaning the data

- Data needs to be very ‘clean’
- ‘1024MB’, ‘1GB’ – convert to same units
- ‘1 CPU’, ‘1 cpu’ – convert to same case
- Base result = ‘0’ – removal of outliers
- ‘1GB’, ‘ 1 GB’, ‘1 GB’ – removal of spurious spaces
- ’1GB’, ‘2GB’, ‘4GB’ – make categorical

- Our reproducibility package contributes code to clean the SPEC CPU 2017 data to support further analyses.

Removal of highly correlated features

- Highly correlated features don’t help with producing better results
- And sometimes make things worse
- Kendall’s rank correlation used to identify those features > 70% corelated with others
- 7 features removed

Pearson and Spearman gave very similar results

Outline

- Predicting Deep Learning execution time
  - Problem and Approach
  - Predicting Execution time for individual operations
  - Predicting for a full model
- Predicting SPEC 2017 performance
  - Problem and Approach
  - The Data
  - Data preparation
  - Deep Learning models
  - Results
- Conclusions

Challenges

- Choosing the best Neural Network isn’t trivial
- Shape of the network
  - Layers and width
- Types of ‘neurons’
- Activation functions
- Loss function
- Optimizers
- Stride size

Challenges

- Choosing the best Neural Network isn't trivial
- Shape of the network
    - Layers and width
- Types of 'neurons'

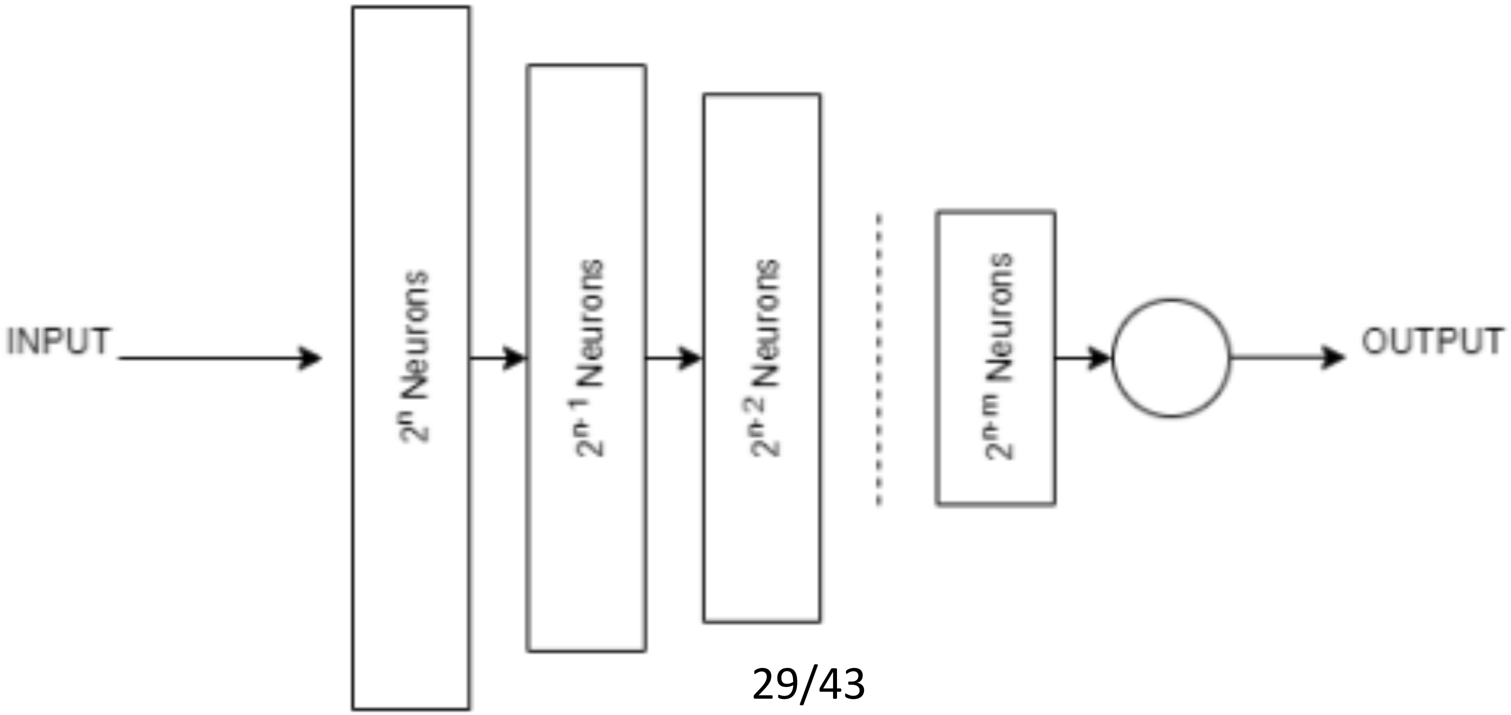
Neural architecture search space

- Activation functions
- Loss function
- Optimizers
- Stride size
- Epochs

Hyperparameter search space

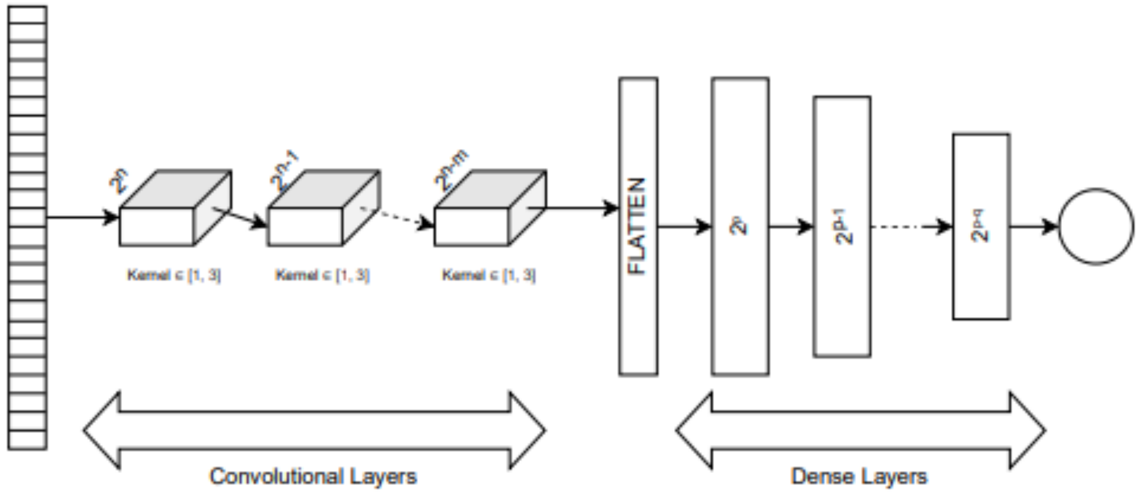
Searching for Neural Network (MLP)

- Fully-Connected Networks: trapezium shaped
  - Number of neurons: From 2^n to 2^{n-m}
    - Range = $n \in [4, \ldots, 11]$, $m \in [1, \ldots, 10]$



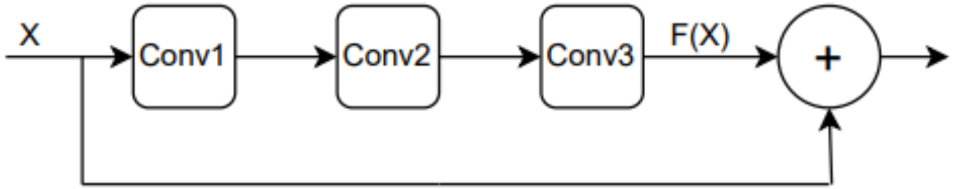
Searching for Neural Network cont. (CNN)

- CNN design: trapezium shaped
  - Number of convolutional layers: From 2^n to 2^{n-m}
    - Range = $n \in [7, \ldots, 11]$, $m \in [4, \ldots, 7]$
    - Kernel $\in [1, 3]$
  - Number of neurons: From 2^p to 2^{p-q}
    - Range = $p \in [7, \ldots, 11]$, $q \in [5, \ldots, 7]$

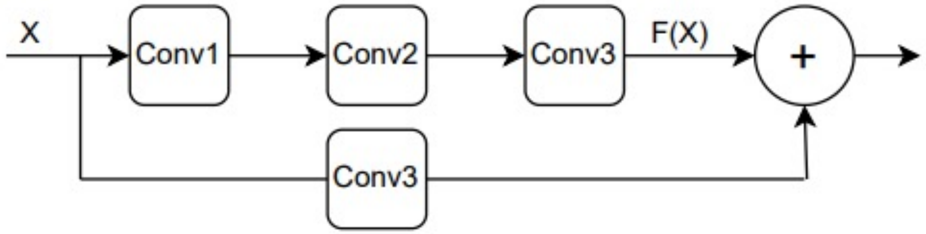


Searching for Neural Network cont. (ResNet Inspired)

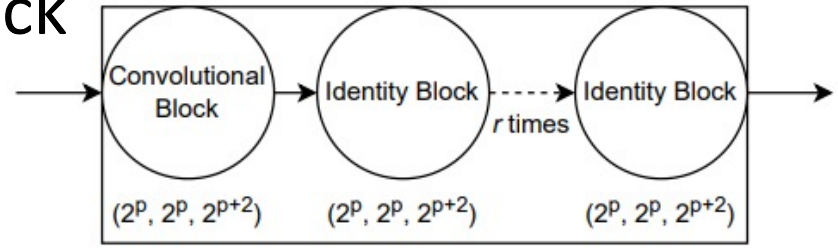
- Identity block



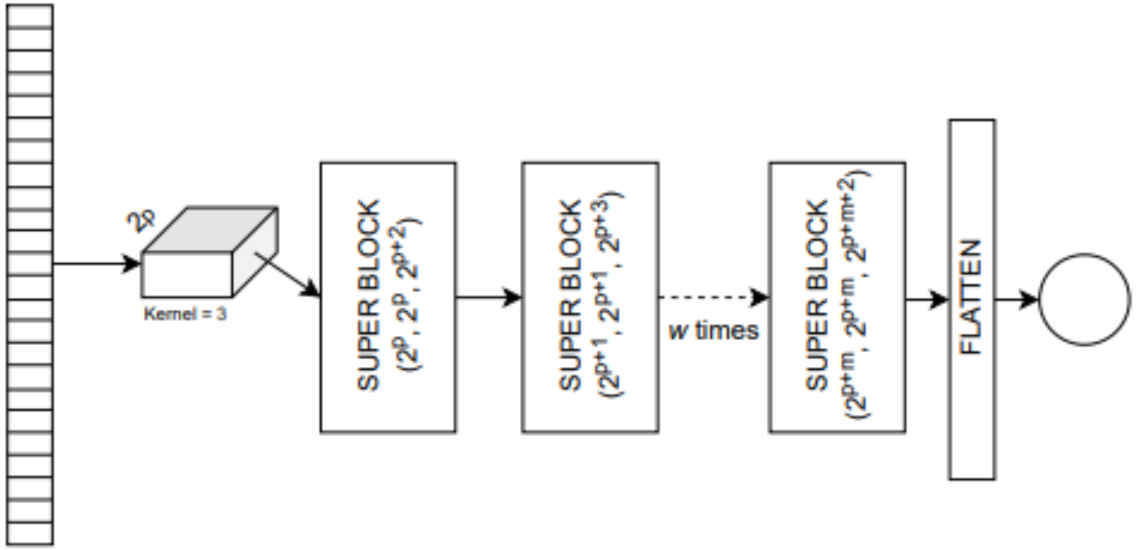
- Convolutional block



- Super block



- Final architecture



Hyperparameter search

- Optimizers: SGD, Adam, Rmsprop
- Loss functions: MAE and MSE
- Activation functions: sigmoid, tanh, ReLU
- Stride size $\in [1, \ldots, 4]$
- Training epochs [50, 100, 150, 200, 250, 300, 350, 400]

Outline

- Predicting Deep Learning execution time
  - Problem and Approach
  - Predicting Execution time for individual operations
  - Predicting for a full model
- Predicting SPEC 2017 performance
  - Problem and Approach
  - The Data
  - Data preparation
  - Deep Learning models
  - Results
- Conclusions

Metrics

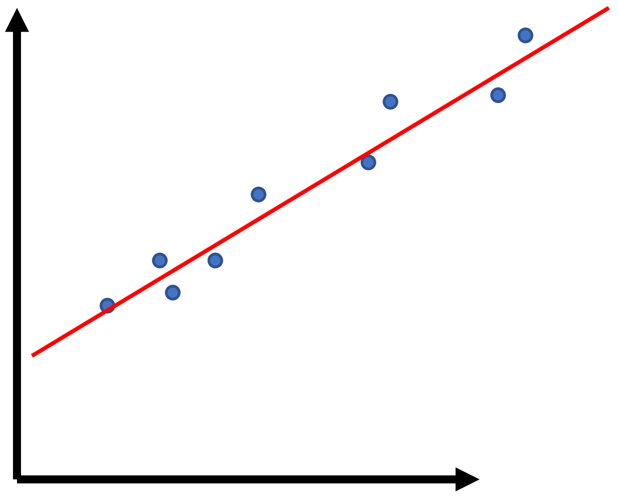
- R^2: strength of the relationship between predictions and actual
  - Closer to 1 is better
- MAE: how big error is between predicted and actual
  - Closer to 0 is better
- MSE: Similar to MAE but more impact from large differences
  - Closer to 0 is better

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - y_i')^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i' - y| \quad MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i' - y_i)^2$$

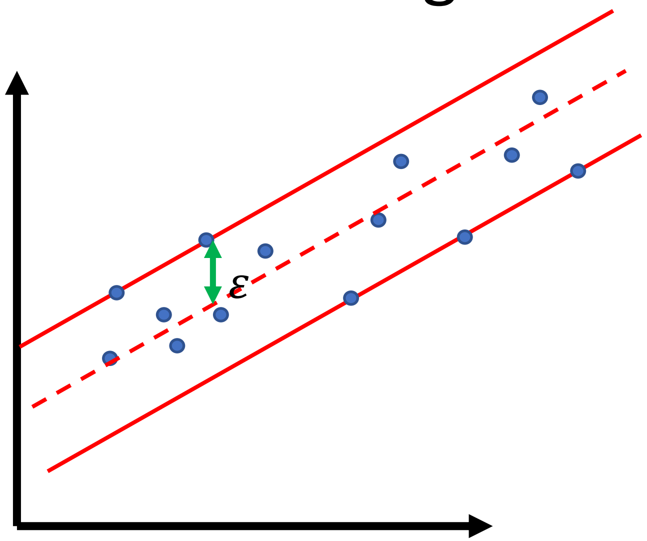
y_i is the true value, y_i' is the predicted value and $\bar{y}_i$ is the mean of all true values

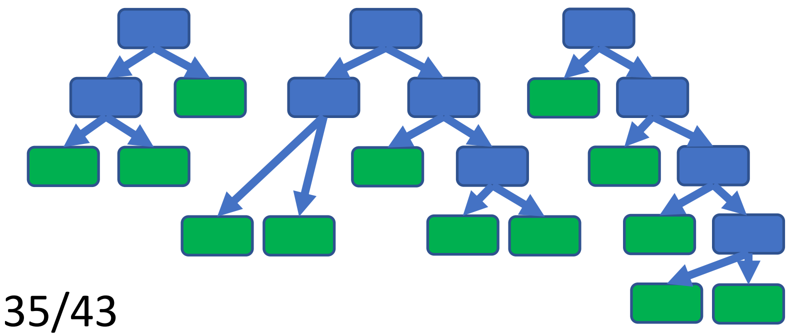
Baseline comparison methods

- Linear Regression



- Support Vector Regression
- Random Forest Regression





Comparison of approaches

Model	Best R^2
Trapezium CNN	0.9864
Random Forest Regression	0.9830
Fully Connected MLP	0.9735
Residual Neural Network	0.9501
Linear Regression	0.5260
Support Vector Regression	-0.0040

Overall Comparison – sorter by R^2

#	Architecture	Loss Fn	Kernel Sizes	Stride Sizes	Number of Filters (m, n)	Neurons in Layers (p, q)	Optimizer	Epochs	R2	MAE	MSE
1	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	250	0.98638701	5.67389728	465.3285655
2	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	250	0.98590661	5.83946465	476.0394343
3	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	Adam	300	0.98579341	5.76197731	494.124225
4	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	150	0.98529142	6.25318407	513.9629513
5	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	RmsProp	150	0.98282719	7.14056732	620.2982421
6	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	200	0.98280914	6.03564805	582.3068145
7	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	300	0.98278342	5.61076184	582.0247239
8	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	300	0.98107176	5.78137347	645.4129883
9	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	RmsProp	250	0.98095925	6.72097815	669.8856237
10	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	200	0.98089907	6.32291809	665.1641919
11	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	Adam	150	0.98047251	6.71537772	663.7030719
12	TriCNN	MAE	3	1	(7, 6, 5, 4)	[9, …, 5]	RmsProp	300	0.98038864	6.9974749	653.5821786
~	RF								0.9803076	4.76701531	688.0001262
13	TriCNN	MAE	3	1	(7, 6, 5, 4)	[9, …, 5]	RmsProp	200	0.98002879	7.62788323	684.7595471
14	TriCNN	MAE	2	1	(9, 7)	[11, …, 6]	Adam	150	0.9793459	6.519971	703.0615545
15	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	Adam	100	0.97782539	8.23651529	754.5381605
16	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	100	0.97748578	7.30871799	757.4994833
17	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	150	0.97726148	6.65855022	772.0747562
18	TriCNN	MAE	3	1	(7, 6, 5, 4)	[9, …, 5]	RmsProp	250	0.97665471	7.86703389	775.8960386
19	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	RmsProp	250	0.97650919	7.97325412	852.3545636
20	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	RmsProp	300	0.97636563	6.91501173	816.7881606
45	TriMLP	MAE				[11, …, 6]	Adam	250	0.97347275	9.12443258	906.1439402
159	Residual	MAE	Number of Superblocks = (2, 5, 5, 2)		((6, 6, 8), (7, 7, 9), (8, 8, 10), (9, 9, 11))	1	RmsProp	250	0.95007233	10.595069	1006.134564
~	LR								0.52639158	82.4596122	15761.16107
~	SVR								-0.0045634	113.749207	33448.30886

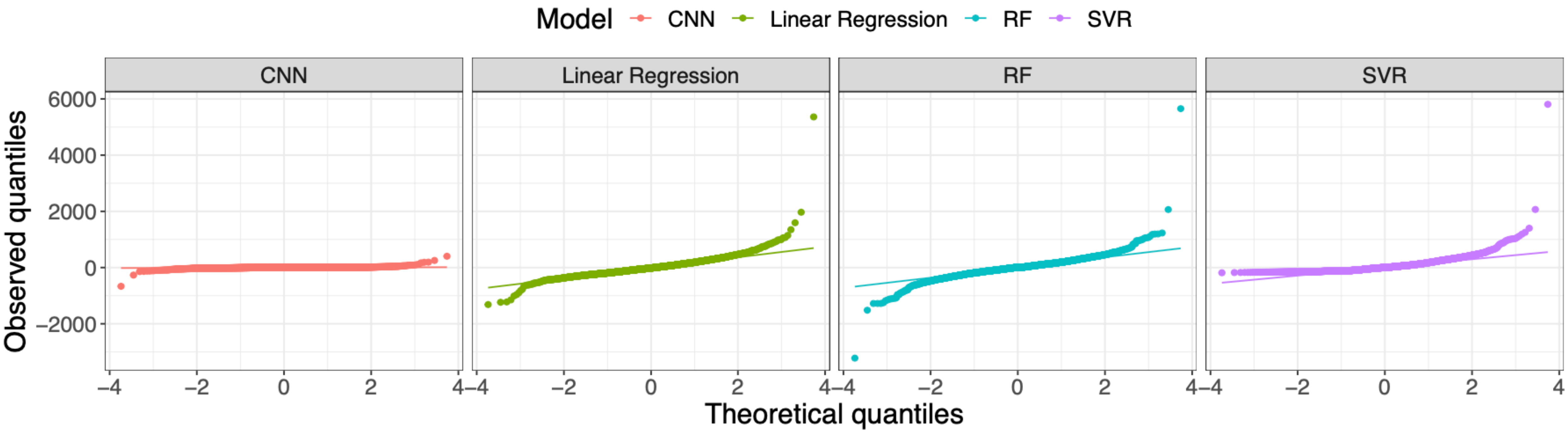
* TriCNN = Trapezium-shaped CNN, RF = Random Forest Regression, TriMLP = Trapezium-shaped MPL, LR = Linear Regression, SVR = Support Vector Regression

Overall Comparison – sorted by MAE

#	Architecture	Loss Fn	Kernel Sizes	Stride Sizes	Number of Filters (m, n)	Neurons in Layers (p, q)	Optimizer	Epochs	R2	MAE	MSE
1	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	250	0.98638701	5.67389728	465.3285655
2	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	250	0.98590661	5.83946465	476.0394343
3	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	Adam	300	0.98579341	5.76197731	494.124225
4	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	150	0.98529142	6.25318407	513.9629513
5	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	300	0.98278342	5.61076184	582.0247239
6	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	200	0.98280914	6.03564805	582.3068145
7	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	RmsProp	150	0.98282719	7.14056732	620.2982421
8	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	300	0.98107176	5.78137347	645.4129883
9	TriCNN	MAE	3	1	(7, 6, 5, 4)	[9, …, 5]	RmsProp	300	0.98038864	6.9974749	653.5821786
10	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	Adam	150	0.98047251	6.71537772	663.7030719
11	TriCNN	MAE	3	1	(9, 7)	[9, …, 5]	Adam	200	0.98089907	6.32291809	665.1641919
12	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	RmsProp	250	0.98095925	6.72097815	669.8856237
13	TriCNN	MAE	3	1	(7, 6, 5, 4)	[9, …, 5]	RmsProp	200	0.98002879	7.62788323	684.7595471
~	RF								0.9803076	4.76701531	688.0001262
14	TriCNN	MAE	2	1	(9, 7)	[11, …, 6]	Adam	150	0.9793459	6.519971	703.0615545
15	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	Adam	100	0.97782539	8.23651529	754.5381605
16	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	100	0.97748578	7.30871799	757.4994833
17	TriCNN	MAE	3	2	(9, 7, 6, 5, 4)	[9, …, 4]	Adam	150	0.97726148	6.65855022	772.0747562
18	TriCNN	MAE	3	1	(7, 6, 5, 4)	[9, …, 5]	RmsProp	250	0.97665471	7.86703389	775.8960386
19	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	RmsProp	200	0.97613855	7.72461632	807.1294185
20	TriCNN	MAE	3	2	(9, 7)	[9, …, 5]	RmsProp	300	0.97636563	6.91501173	816.7881606
48	TriMLP	MAE				[11, …, 6]	Adam	250	0.97347275	9.12443258	906.1439402
135	Residual	MAE	Number of Superblocks = (2, 5, 5, 2)		((6, 6, 8), (7, 7, 9), (8, 8, 10), (9, 9, 11))	1	RmsProp	250	0.95007233	10.595069	1006.134564
	LR								0.52639158	82.4596122	15761.16107
	SVR								-0.0045634	113.749207	33448.30886

* TriCNN = Trapezium-shaped CNN, RF = Random Forest Regression, TriMLP = Trapezium-shaped MPL, LR = Linear Regression, SVR = Support Vector Regression

How do we do across the range?

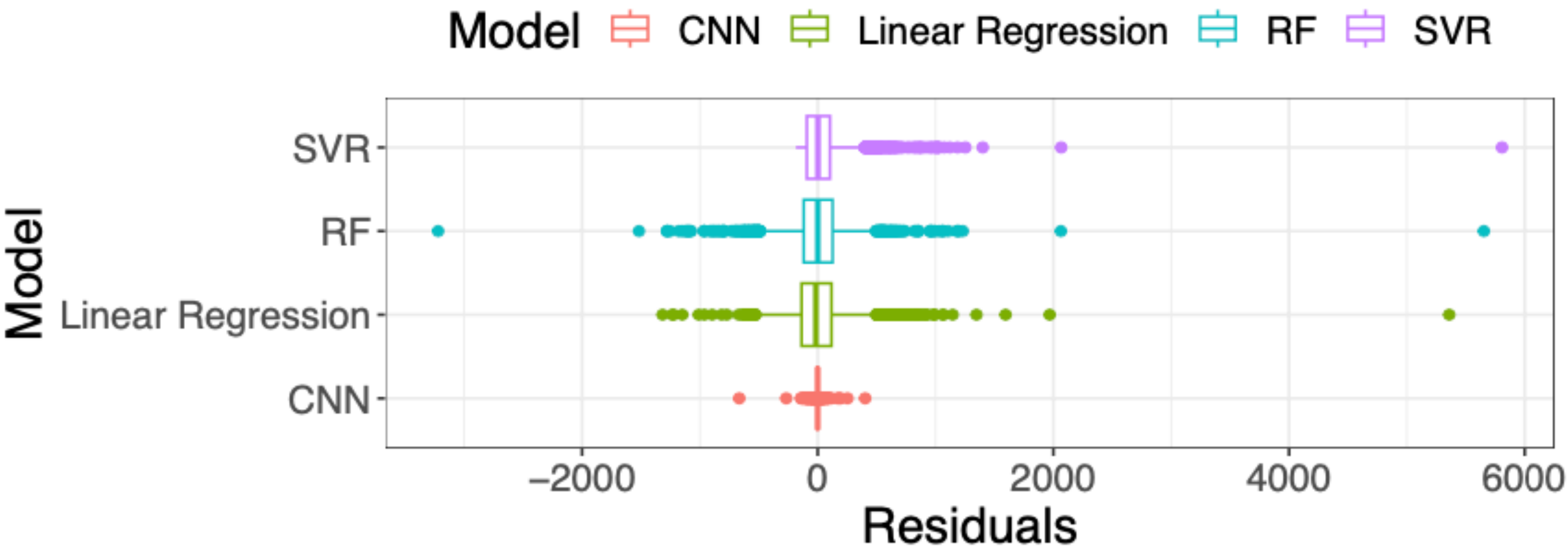


Hyperparameters

- Optimizer: Adam though RMSprop close
- Loss function: MAE, even when metric was MSE
- Activation function: Sigmoid
- Stride: 1 or 2
- Kernel size: normally 3
- Training epochs: Normally 250, though some exceptions

- Though model dependent

Residuals of different models



Outline

- Predicting Deep Learning execution time
  - Problem and Approach
  - Predicting Execution time for individual operations
  - Predicting for a full model
- Predicting SPEC 2017 performance
  - Problem and Approach
  - The Data
  - Data preparation
  - Deep Learning models
  - Results
- Conclusions

Conclusions

- Demonstrated we can accurately predict the performance of systems by using Deep Learning models – both DL models and SPEC2017
- DL outperforms Linear approaches – often by a lot
- (1D) CNNs work the best, ResNet doesn't work well here
- A Neural Architecture Search reveals improvements in results
  - More searching could improve, but diminishing returns
- Random Forest is a reasonable alternative – for main cases
- More features would improve performance – break down differences

stephen.mcgough@newcastle.ac.uk

https://github.com/CDECatapult/ml-performance-prediction https://github.com/cengizmehmet/BenchmarkNets

With thanks to: Mehmet Cengiz, Matthew Forshaw, Amir Atapour-Abarghouei, Stephen Bonner, Daniel Justus and John Brennan