Predicting the execution time of secure neural network inference: Comprehensive Analysis and Results

Eloise Zhang and Zoltán Ádám Mann

University of Amsterdam, Amsterdam, The Netherlands

Abstract. In the secure neural network inference (SNNI) problem, a service provider offers inference as a service with a pre-trained neural network (NN). Clients can use the service by providing an input and obtaining the output of the inference with the NN. For reasons of privacy and intellectual property protection, the service provider must not learn anything about the input or the output, and the client must not learn anything about the internal parameters of the NN. This is possible by applying techniques like multi-party computing (MPC) or homomorphic encryption (HE), although with a significant performance overhead.
One way to improve the efficiency of SNNI is by selecting NN architectures that can be evaluated faster using MPC or HE. For this, it would be important to predict how long SNNI with a given NN takes. This turns out to be challenging. Traditional predictors for NN inference time, like the number of parameters in the NN, are poor predictors of SNNI execution time, since they ignore the characteristics of cryptographic protocols. This paper is the first to address this problem. We propose three different prediction methods for SNNI execution time, and investigate experimentally their strengths and weaknesses. The results show that the proposed methods offer different advantages in terms of accuracy and speed.

Here is an extended overview of the results from the paper "Predicting the Execution Time of Secure Neural Network Inference". For a detailed understanding of the research background and construction of prediction models, we kindly refer readers to the original paper. In this extended discussion, we delve deeper into our findings, offering a comprehensive analysis of execution time prediction.

1 Experiments

1.1 Process of running experiments

Our experiments involve three phases: data generation, model tuning and training, and evaluation (see Fig. 1). The subsequent paragraphs give details of each phase. The data and scripts of our experiments are publicly available[1].

[1] https://github.com/Eloise2000/SNNI-Performance-Evaluator

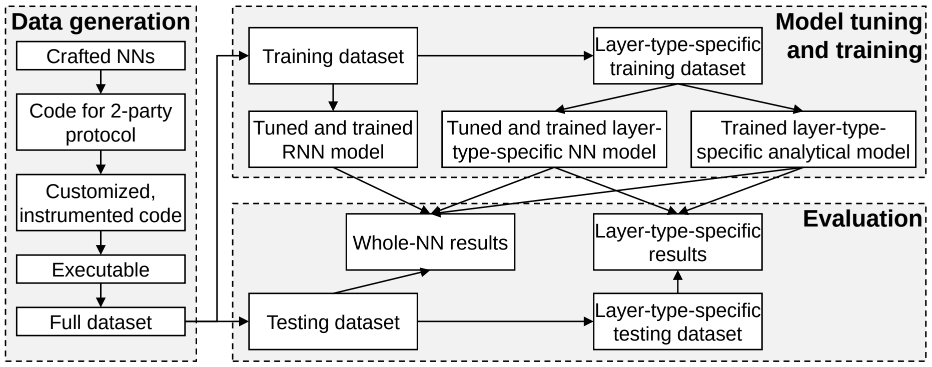
Data generation
Crafted NNs
Code for 2-party protocol
Customized, instrumented code
Executable
Full dataset
Model tuning and training
Training dataset
Layer-type-specific training dataset
Tuned and trained RNN model
Tuned and trained layer-type-specific NN model
Trained layer-type-specific analytical model
Evaluation
Whole-NN results
Layer-type-specific results
Testing dataset
Layer-type-specific testing dataset

Fig. 1. Overview of the experimentation process

Data generation. To train our prediction models, we first acquire data of SNNI execution time for various layer sizes for each layer type. This process entails the creation of comprehensive NNs for profiling SNNI execution time. When crafting these NNs, we deliberately diversify their composition by incorporating layers with varying configurations. This provides us with a richer dataset in a given amount of SNNI time.

To run SNNI on the created NNs, we first compile them into code for the appropriate protocol, using a configuration file. The code is further customized to align with a specific monitoring protocol. Then, we instrument the code to track runtime and the influencing factors for each layer. Afterwards, we compile and link the code to generate an executable for SNNI. Following this preparatory phase, we execute the secure protocol on the NNs and capture the relevant metrics. The result is a comprehensive dataset containing layer-specific descriptions of the neural network architecture, along with the corresponding execution times. We split this dataset, allocating 80% of it for training, and the remaining 20% for testing purposes.

Model tuning and training. From the training dataset, we construct layer-type-specific datasets suitable for the per-layer prediction approaches. This involves extracting the layer-specific data from the dataset generated by SNNI, grouping the layer data by type (e.g., CONV, ReLU layers), and then splitting the data into features, representing the characteristics of each layer, and labels, representing the time taken to execute each layer.

Models are trained for each method from the paper method. The RNN is trained using the whole training dataset. For training the analytical models and NNs per layer type, the corresponding layer-type-specific training datasets are used.

To enhance the performance of the NN and RNN models, we employ a hyperparameter tuning strategy. This involves conducting multiple trials with the RandomSearch tuner from the Keras Tuner library [6] to explore a range of hyperparameters, including learning rate, batch size, and network architecture parameters. After each trial, we extract the hyperparameters that result in the lowest validation loss, ensuring optimal model performance.

The complexity of the search space for building NN models varies across different layer types; for example, more complex layers may require a higher number of hidden layers in the prediction NNs. To maintain computational efficiency, we limit the number of hidden layers to a maximum of 5. For further details on the investigated values for three different settings (local, LAN, and WAN), please refer to the "Investigated range" section of Table 1-3. In total, we conducted 20 trials for each layer, exploring various hyperparameter combinations. Each trial included training with early stopping based on validation loss to prevent overfitting. The hyperparameters yielding the lowest validation loss were selected and are presented in Table 1-3 under the "Chosen value" column.

For constructing RNN models, we conducted 30 trials to identify the optimal hyperparameter configuration, utilizing the mean squared error as the loss function. The range of explored values and the selected configurations are outlined in Tables 4-6.

Lastly, we conduct model fitting for all three models, fine-tuning the model parameters to achieve the best fit with the training data and minimize prediction errors.

Evaluation. As with the training dataset, also the testing dataset is regrouped into a set of layer-type-specific datasets.

We assess the effectiveness of our prediction methods through a comprehensive evaluation process. For the per-layer prediction approaches, we conduct individual layer testing. This involves evaluating the prediction performance for each type of layer independently, using the layer-type-specific testing datasets. In addition, we assess the prediction performance of all methods on complete NNs, using the complete testing dataset. The detailed results of these evaluations are presented in Section 1.3.

1.2 Experimental setup

We use the Easy Secure Multi-Party Computation (EzPC) project [1] and Cheetah [5] for executing SNNI on custom NNs. We employ the Athos compiler to compile our tensorflow2-based NNs into code compatible with SNNI.

The input dimensions are fixed to 224x224x3, a common image size in the ImageNet dataset. For CONV, MP, and AvP layers, we opted for square kernels with odd width (1, 3, 5, 7), a prevalent practice. As possible values for the number of filters, we used powers of 2, ranging from 16 to 1024.

For execution, both model weights and input data are converted from floating point to fixed point representation. We use a bitlength of 41 and a precision of 12 bits, a common and widely used configuration [7].

Our prediction models' efficacy is evaluated on two popular NNs: SqueezeNet, ResNet-50, and DenseNet-121. SqueezeNet is a compact deep learning architecture designed for efficient image classification, achieving high accuracy with a significantly reduced model size [4]. ResNet-50, a member of the ResNet family, is known for its effectiveness in training very deep networks [2]. DenseNet-121,

Table 1. NN hyperparameters for the Local experiment. For learning rate, the investigated range was 10^{-4}–10^{-2} in all cases. C=client, S=server, st=step.

SNNI layer	Investigated range		Chosen value		
	hidden layers	neurons per layer	hidden layers	neurons per layer	learning rate
FC C	2–5	32–512 (st 32)	5	192, 448, 224, 64, 416	$2.16 \cdot 10^{-3}$
FC S	2–5	32–512 (st 32)	4	416, 448, 384, 448	$6.59 \cdot 10^{-3}$
CONV C	1–4	32–512 (st 32)	4	288, 96, 416, 320	$6.34 \cdot 10^{-3}$
CONV S	1–4	32–512 (st 32)	3	352, 352, 32	$9.90 \cdot 10^{-3}$
MP C	1–5	32–512 (st 32)	5	512, 320, 192, 96, 512	$5.79 \cdot 10^{-3}$
MP S	1–5	32–512 (st 32)	4	448, 192, 480, 32	$9.70 \cdot 10^{-3}$
AvP C	1–5	32–512 (st 32)	5	256, 416, 32, 32, 32	$9.84 \cdot 10^{-3}$
AvP S	1–5	32–512 (st 32)	5	128, 288, 448, 32, 224	$4.94 \cdot 10^{-4}$
ReLU C	1–4	32–256 (st 32)	2	96, 32	$2.96 \cdot 10^{-3}$
ReLU S	1–4	32–256 (st 32)	3	160, 32, 32	$9.31 \cdot 10^{-3}$
BN C	1–4	32–256 (st 32)	4	224, 256, 128, 64	$4.15 \cdot 10^{-3}$
BN S	1–4	32–256 (st 32)	4	160, 96, 256, 64	$6.96 \cdot 10^{-3}$

characterized by dense connections between layers, offers advantages in feature propagation and parameter efficiency [3].

Our experiments are conducted on two virtual machines, each equipped with 8 gigabytes of RAM. These experiments are carried out in three distinct technical settings. In the *Local* setting, the client and server processes run on the same machine. In the *LAN* configuration, the server and client processes operate on separate machines, interconnected via a local network. Additionally, we simulate a *WAN* setting using the Linux Traffic Control (TC) subsystem. The bandwidth between the server and client is approximately 7 Gbits/s for LAN and 380 Mbits/s for WAN. The round-trip times are approximately 0.5ms for LAN and 10ms for WAN, respectively.

1.3 Results

Per-layer comparison Here, we present the results of the first two prediction methods – analytical model and NN-based – for different types of layers (see Table 8). Examining the coefficient of determination (R2) scores, we observe that both methods achieve relatively high R2 values, indicating a strong correlation between predicted and actual execution times. In most cases, the NN approach leads to more accurate predictions than the analytical approach, as evidenced by higher R2 scores and lower MAE (Mean Absolute Error). This is particularly true for CONV and AvP layers, where the NN approach could be up to 77% better than the prediction of the analytical model. On average, the NN model exhibits a runtime prediction accuracy improvement of approximately 115 ms per layer. However, for FC layers, the analytical approach demonstrates comparable or slightly superior predictions. This is attributed to both the analytical model

Table 2. NN hyperparameters for the LAN experiment. For learning rate, the investigated range was 10^{-4}–10^{-2} in all cases. C=client, S=server, st=step.

SNNI layer	Investigated range		Chosen value		
	hidden layers	neurons per layer	hidden layers	neurons per layer	learning rate
FC C	3–4	16–512 (st 16)	4	96, 208, 512, 384	$3.64 \cdot 10^{-3}$
FC S	3–4	16–512 (st 16)	4	16, 384, 48, 512	$8.08 \cdot 10^{-3}$
CONV C	3–4	16–640 (st 16)	4	112, 320, 432, 304	$3.55 \cdot 10^{-3}$
CONV S	3–4	16–640 (st 16)	4	32, 352, 320, 560	$8.81 \cdot 10^{-3}$
MP C	3–4	16–640 (st 16)	4	272, 640, 352, 256	$4.26 \cdot 10^{-4}$
MP S	3–4	16–640 (st 16)	4	432, 448, 560, 416	$8.02 \cdot 10^{-3}$
AvP C	3–4	16–640 (st 16)	4	448, 464, 352, 416	$5.49 \cdot 10^{-3}$
AvP S	3–4	16–640 (st 16)	4	368, 384, 288, 512	$9.38 \cdot 10^{-4}$
ReLU C	2–4	16–512 (st 16)	4	256, 416, 32, 352	$5.87 \cdot 10^{-3}$
ReLU S	2–4	16–512 (st 16)	3	352, 32, 336	$2.65 \cdot 10^{-3}$
BN C	3–4	16–512 (st 16)	4	416, 480, 240, 512	$9.80 \cdot 10^{-3}$
BN S	3–4	16–512 (st 16)	4	448, 304, 176, 304	$2.29 \cdot 10^{-3}$

and NN approach utilizing the same raw features. Given our limitation on the depth of NN layers to prevent excessively long prediction times, the analytical model, which incorporates more meticulously designed features, can yield better outcomes in some cases.

On the other hand, the NN method incurs higher prediction times. This trade-off between prediction accuracy and computational cost should be taken into consideration when selecting an appropriate method for specific layer types in SNNI. It is worth mentioning that even with the NN approach, the prediction time remains relatively low, consistently under 0.2 seconds. In particular, the prediction time is always lower – and in most cases significantly lower – than the average SNNI execution time. This quick response time allows clients to efficiently select SNNI services based on predicted inference durations.

Whole-NN comparison Here we present experiments conducted on entire NNs using all three prediction methods. The results are summarized in Table 10. The analytical model exhibited an error rate of approximately 20%, while the NN approach demonstrated exceptional precision with a mere 5% error, underscoring its remarkable predictive accuracy. Surpassing this, the RNN approach showcased an even lower average error of only 2.1%, resulting in a negligible 4-second deviation for an SNNI of 231 seconds.

The RNN approach consistently outperforms the other approaches in accuracy. Conversely, the analytical model excels in speed. Thus, there is a clear trade-off between accuracy and prediction time. Employing the NN approach, for instance, would entail spending approximately 50 times more time on the prediction model than with the analytical model. For RNN, this figure rises to about 150 times. Nevertheless, even with the slowest RNN model, the prediction

Table 3. NN hyperparameters for the WAN experiment. For learning rate, the investigated range was 10^{-4}–10^{-2} in all cases. C=client, S=server, st=step.

SNNI layer	Investigated range		Chosen value		
	hidden layers	neurons per layer	hidden layers	neurons per layer	learning rate
FC C	3–4	16–512 (st 16)	3	368, 208, 160	$7.49 \cdot 10^{-4}$
FC S	3–4	16–512 (st 16)	3	576, 80, 144	$3.79 \cdot 10^{-3}$
CONV C	3–4	16–640 (st 16)	3	368, 336, 480	$7.11 \cdot 10^{-3}$
CONV S	3–4	16–640 (st 16)	4	368, 640, 432, 224	$6.39 \cdot 10^{-3}$
MP C	3–4	16–640 (st 16)	4	560, 544, 224, 592	$3.14 \cdot 10^{-4}$
MP S	3–4	16–640 (st 16)	4	512, 288, 624, 432	$8.00 \cdot 10^{-3}$
AvP C	3–4	16–640 (st 16)	4	144, 512, 144, 288	$7.23 \cdot 10^{-3}$
AvP S	3–4	16–640 (st 16)	3	560, 64, 528	$1.89 \cdot 10^{-4}$
ReLU C	2–4	16–512 (st 16)	4	240, 96, 32, 80	$2.91 \cdot 10^{-3}$
ReLU S	2–4	16–512 (st 16)	4	416, 64, 208, 400	$4.55 \cdot 10^{-3}$
BN C	3–4	16–512 (st 16)	4	368, 320, 128, 448	$1.27 \cdot 10^{-3}$
BN S	3–4	16–512 (st 16)	4	192, 464, 176, 448	$2.61 \cdot 10^{-3}$

Table 4. RNN hyperparameters for the Local experiment

SNNI role	Investigated range			Chosen value		
	hidden layers	neurons per layer	learning rate	hidden layers	neurons per layer	learning rate
Client	6–12	32–512 (step 32)	10^{-5}–10^{-2}	12	96, 128, 352, 448, 128, 96, 512, 384, 192, 384, 384, 256	$4.17 \cdot 10^{-3}$
Server	6–12	32–512 (step 32)	10^{-5}–10^{-2}	10	256, 192, 192, 320, 256, 32, 352, 416, 32, 288	$3.53 \cdot 10^{-3}$

time only amounts to roughly 3 seconds—significantly less than the SNNI time, making it practical and feasible for use.

References

1. Ezpc: Easy secure multi-party computation. `https://github.com/mpc-msri/EzPC`
2. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
3. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4700–4708 (2017)
4. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016)
5. Lab, A.G.: Cheetah: Lean and fast secure two-party deep neural network inference. `https://github.com/Alibaba-Gemini-Lab/OpenCheetah`

Table 5. RNN hyperparameters for the LAN experiment

SNNI role	Investigated range			Chosen value		
	hidden layers	neurons per layer	learning rate	hidden layers	neurons per layer	learning rate
Client	6–20	32–512 (step 32)	10^{-5}–10^{-2}	11	480, 256, 288, 128, 352, 384, 160, 416, 480, 352, 32	$6.93 \cdot 10^{-4}$
Server	6–12	32–512 (step 32)	10^{-5}–10^{-2}	11	96, 128, 32, 192, 128, 96, 416, 352, 480, 512, 416	$1.09 \cdot 10^{-3}$

Table 6. RNN hyperparameters for the WAN experiment

SNNI role	Investigated range			Chosen value		
	hidden layers	neurons per layer	learning rate	hidden layers	neurons per layer	learning rate
Client	6–20	32–512 (step 32)	10^{-5}–10^{-2}	12	512, 320, 96, 352, 64, 224, 352, 256, 480, 416, 32, 96	$8.05 \cdot 10^{-3}$
Server	6–12	32–512 (step 32)	10^{-5}–10^{-2}	19	448, 480, 128, 320, 416, 416, 416, 352, 96, 384, 384, 448, 416, 352, 288, 96, 32, 32	$9.02 \cdot 10^{-3}$

6. O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., et al.: Kerastuner. `https://github.com/keras-team/keras-tuner` (2019)
7. Srinivasan, W.Z., Akshayaram, P., Ada, P.R.: Delphi: A cryptographic inference service for neural networks. In: Proc. 29th USENIX Secur. Symp. pp. 2505–2522 (2019)

Table 7. Results for different types of layers. t_{layer}: average SNNI time for the given type of layer in our dataset. *MAE*: average absolute error of the predicted value. *R2*: coefficient of determination. t_{pred}: average time needed for running the prediction.

		Local experiment				LAN experiment			
Layer	Method	t_{layer} [ms]	MAE [ms]	R2	t_{pred} [ms]	t_{layer} [ms]	MAE [ms]	R2	t_{pred} [ms]
FC C	Analytical	234	82	0.813	4	459	72	0.958	4
	NN		121	0.810	184		105	0.952	153
FC S	Analytical	228	81	0.808	4	448	71	0.957	4
	NN		112	0.817	114		105	0.952	152
CONV C	Analytical	959	415	0.761	4	2549	1402	0.642	4
	NN		41	0.997	144		165	0.989	154
CONV S	Analytical	722	417	0.552	3	2159	1401	0.521	4
	NN		36	0.997	156		73	0.998	154
MP C	Analytical	1267	404	0.932	3	3211	286	0.995	3
	NN		161	0.987	144		296	0.990	149
MP S	Analytical	1328	421	0.937	3	3300	314	0.995	2
	NN		181	0.983	157		221	0.995	158
AvP C	Analytical	272	141	0.554	4	727	111	0.960	3
	NN		104	0.734	135		106	0.965	158
AvP S	Analytical	295	148	0.578	3	766	128	0.954	3
	NN		96	0.740	193		113	0.967	153
ReLU C	Analytical	719	341	0.765	2	1523	202	0.986	2
	NN		302	0.793	123		195	0.985	160
ReLU S	Analytical	880	445	0.748	2	1796	351	0.937	2
	NN		389	0.748	128		303	0.947	145
BN C	Analytical	1290	60	0.906	3	821	94	0.942	2
	NN		42	0.952	151		62	0.976	166
BN S	Analytical	894	58	0.847	3	582	90	0.878	2
	NN		42	0.921	151		53	0.955	146

Table 8. Results for different types of layers. t_{layer}: average SNNI time for the given type of layer in our dataset. *MAE*: average absolute error of the predicted value. *R2*: coefficient of determination. t_{pred}: average time needed for running the prediction.

		LAN experiment				WAN experiment			
Layer	Method	t_{layer} [ms]	MAE [ms]	R2	t_{pred} [ms]	t_{layer} [ms]	MAE [ms]	R2	t_{pred} [ms]
FC C	Analytical	459	72	0.958	4	491	85	0.953	4
	NN		105	0.952	153		83	0.968	139
FC S	Analytical	448	71	0.957	4	472	82	0.953	4
	NN		105	0.952	152		90	0.965	138
CONV C	Analytical	2549	1402	0.642	4	2745	1449	0.646	4
	NN		165	0.989	154		135	0.996	164
CONV S	Analytical	2159	1401	0.521	4	2350	1455	0.544	4
	NN		73	0.998	154		74	0.998	167
MP C	Analytical	3211	286	0.995	3	3757	576	0.992	2
	NN		296	0.990	149		374	0.983	164
MP S	Analytical	3300	314	0.995	2	3855	597	0.992	3
	NN		221	0.995	158		284	0.987	188
AvP C	Analytical	727	111	0.960	3	909	170	0.958	3
	NN		106	0.965	158		132	0.960	152
AvP S	Analytical	766	128	0.954	3	939	182	0.952	3
	NN		113	0.967	153		114	0.971	145
ReLU C	Analytical	1523	202	0.986	2	1655	214	0.985	2
	NN		195	0.985	160		210	0.984	185
ReLU S	Analytical	1796	351	0.937	2	1936	357	0.943	2
	NN		303	0.947	145		343	0.934	175
BN C	Analytical	821	94	0.942	2	1076	163	0.894	2
	NN		62	0.976	166		84	0.948	163
BN S	Analytical	582	90	0.878	2	831	163	0.825	2
	NN		53	0.955	146		86	0.915	170

Table 9. Results of the three prediction methods (*Analytical*, *NN per layer*, *RNN*) for different NNs. *NN* is the NN that SNNI is applied to. t_{SNNI} is the duration of SNNI on the NN. *Error* is given as the absolute error of the predicted value, and as the absolute error divided by the actual value. t_{pred} is the time needed for running the prediction.

		Local experiment				LAN experiment			
NN	Method	t_{SNNI} [s]	Error [s]	Error [%]	t_{pred} [ms]	t_{SNNI} [s]	Error [s]	Error [%]	t_{pred} [ms]
SqueezeNet Client	Analytical	43.0	8.1	18.8	23	104.0	5.5	5.3	21
	NN		3.2	7.4	1195		0.6	0.6	1647
	RNN		0.9	2.1	3831		0.3	0.3	2994
SqueezeNet Server	Analytical	43.0	7.1	16.5	22	104.0	5.1	4.9	22
	NN		2.1	4.9	1160		1.6	1.5	1728
	RNN		0.3	0.7	3449		4.4	4.2	2817
ResNet-50 Client	Analytical	231.2	52.1	22.5	35	559.8	158.7	28.3	35
	NN		13.6	5.9	1752		3.3	0.6	2473
	RNN		8.7	3.8	4073		10.5	1.9	3051
ResNet-50 Server	Analytical	231.2	52.9	22.9	34	559.8	161.1	28.8	38
	NN		10.2	4.4	1600		19.3	3.4	2264
	RNN		4.1	1.8	3169		5.2	0.9	2950

Table 10. Results of the three prediction methods (*Analytical*, *NN per layer*, *RNN*) for different NNs. *NN* is the NN that SNNI is applied to. t_{SNNI} is the duration of SNNI on the NN. *Error* is given as the absolute error of the predicted value, and as the absolute error divided by the actual value. t_{pred} is the time needed for running the prediction.

		LAN experiment				WAN experiment			
NN	Method	t_{SNNI} [s]	Error [s]	Error [%]	t_{pred} [ms]	t_{SNNI} [s]	Error [s]	Error [%]	t_{pred} [ms]
SqueezeNet Client	Analytical		5.5	5.3	21		7.5	6.6	23
	NN	104.0	0.6	0.6	1647	113.7	8.1	7.12	1521
	RNN		0.3	0.3	2994		4.9	4.3	3116
SqueezeNet Server	Analytical		5.1	4.9	22		6.9	6.1	23
	NN	104.0	1.6	1.5	1728	113.7	8.0	7.0	1576
	RNN		4.4	4.2	2817		5.5	4.8	3717
ResNet-50 Client	Analytical		158.7	28.3	35		156.0	26.5	63
	NN	559.8	3.3	0.6	2473	587.7	0.7	0.1	1966
	RNN		10.5	1.9	3051		10.4	1.7	3083
ResNet-50 Server	Analytical		161.1	28.8	38		158.8	27.0	57
	NN	559.8	19.3	3.4	2264	587.7	4.7	0.8	2284
	RNN		5.2	0.9	2950		30.5	5.2	3977
DenseNet-121 Client	Analytical		12.2	2.4	30		8.6	1.6	31
	NN	504.0	6.7	1.3	2274	546.7	12.1	2.2	1804
	RNN		12.0	2.4	3427		6.5	1.2	2999
DenseNet-121 Server	Analytical		6.1	1.2	30		3.5	0.64	30
	NN	504.0	18.0	3.6	2217	546.7	30.7	5.6	2051
	RNN		20.4	4.0	2985		21.6	3.9	3881