Examination of stacked machine learning models for predicting coefficient of drag on non-spherical particles in low reynolds number flows

1st Benjamin Elm Jonsson
Computational Engineering and Design
Fraunhofer Chalmers Centre
Gothenburg, Sweden
benjamin.elm.jonsson@fcc.chalmers.se

2nd Niclas Persson
Computational Engineering and Design
Fraunhofer Chalmers Centre
Gothenburg, Sweden
niclas.persson@fcc.chalmers.se

***Abstract*—In this study the effectiveness of combining deep neural networks (DNN) and convolutional neural networks (CNN) in a stacking structure for predicting the coefficient of drag for small non-spherical object in low reynolds number gas-solid flow. The particles are randomly generated and the flow is solved using an in-house numerical solver. The results show that using a CNN as a way of including the geometrical properties of the particle does improve the ability to make prediction compared to using only a DNN with tabular data. As a baseline comparison a Support vector regression (SVR) was implemented to also compare the results with less complex methods. Whilst SVR proved to perform quite well the proposed stacked model showed significant improvement. One benefit of the stacked model where several networks make a prediction and then a final network use their prediction to make a final prediction is the modularity. One can quite easily add more networks or empirical correlations.**

Keywords: Stacked neural network; Predict coefficient of drag; non-uniform particles; Support vector regression;

NOMENCLATURE

ρ	Fluid density $[kg/m3]$
p	Stress $[Pa]$
μ	Kinematic Viscosity $[m^2/s]$
u	Fluid Velocity $[m/s]$
L_c	Characteristic length [m]
A_p	Projected area, YZ-plane $[m^2]$
Re	Reynolds number [1]
F	Force $[N]$
f	Body acceleractions $[N]$
r	Radius [m]
$\hat{n}$	Surface-normal unit vector [1]

I. INTRODUCTION.

Using machine learning in combination with fluid mechanics is a prosporous field with the number of publications rising rapidely. In many cases different machine learning techniques are being utelized to predict constants or model different variables in increasingly complex scenarios. An example of this is using a stacked neural network to predict the drag coefficient for different shapes [1]. In this specific task of predicting the drag coefficient there are many ways to execute the task and many varieties to take in to account. The previously mentioned study chose to examine how a neural network and empirical correlations could predict the drag coefficient when faced with particles of different shapes. Much research has been performed on non-spherical objects, though often they are limited to uniform or regular-shaped particles [2]–[7]. This is closely related to the topic of this report, however, this report instead consideres highly non-uniform particles . This is to simulate a real soot particle which does not have a single unique structure. Instead, the structure of a soot particle is more or less random. These more complex geometry might deviate a lot more from the expected drag coefficient, C_d, of a spherical particle and are harder to predict. Furthermore it is challenging to capture and describe the geometries in relations to the flow as tabular data, which is why in this report a convolutional neural network was used to learn how a certain geometry impacts the C_d. Many times objects or particles are assumed to be perfectly spherical, but it has ben shown that particles shape heavily impacts

the drag-force in a flow [8], and in most applications the particles are generally non-spherical or irregural spherical, such as soot, biomass and sand [9].

For this study a neural-network structure is created and trained to predict the coefficient of drag C_d. This is done by using different models, with a single deep neural network, with a stacked network consisting of 2 deep neural networks and a convolutional neural network and finally it is done with support vector regression as a baseline to compare performance with. These tools are used in efforts to optimize simulations and save time [10].

In order to accomplish this data is needed for training, validation and finally testing. To generate this data, particles of various sizes and shapes are generated to be simulated over. From this simulation data is generated that is later saved.

A. Motivation.

The main motivation for this study is to save computational power by utlizing machine learning models to replace or aid fluid simulations. This will free up both computational time such that more simulations can be performed and create more data which can be used. As a step in this direction the performance for the proposed structures is evaluated when tasked with predicting the coefficient of drag, C_d, for a simple flow.

B. Scope

This report and its results are limited in serveral ways. First off the particles used for simulations are considered random, but is not completely so. The particles consists of a number of sub-spheres with a fixed radius, $r = 0.5 \cdot 10^{-3}$ which are placed in a discrete fashion with respect to each other. Explicitly, this corresponds to the new sphere choosing either forwards, backwards, right, left, up or down with respect to another sphere in the particle. These subspheres can be placed with either no overlap or an overlap of one radius, with equal probability. The number of subspheres in each generated particle varies from one to a maximum of 15 with a uniform distibution. This limits the size and complexity of the simulations, also limiting the correspondence with reality. In total the dataset used consist of 627 datapoints and images in each plane, which is a limiting factor and is considered quite low, especially when used in convolution neural network.

II. Theory

This part of the report will discuss and explain the theory behind the contents of the report. The theory itself is split in to two main parts, fluid mechanics and machine learning. Furthermore, the machine learning part is split up between neural networks and support vector regression since these are distinctly different.

A. Fluid Mechanics.

The solver used is $IBOflow$ (immersed boundary octree flow solver) which is a multiphase solver built in-house by Fraunhofer Chalmers Centre, FCC. In our case it solves the continuity equation as well as the momentum equations,

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{1}$$

$$\rho \frac{\partial u_i}{\partial t} + \rho u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu \frac{\partial u_i}{\partial x_j}\right) + f_i, \tag{2}$$

where ρ is the density of the fluid,u_i is the velocity field, p is the pressure, μ is the kinematic viscosity and f_i is a body force. This gives the pressure field needed to calculate the resulting forces on the particles. This is calculated as the surface integral of the pressure field over the surface of the particles,

$$F = -\int_S p \cdot \hat{n} dS, \tag{3}$$

where S denotes a surface, $\hat{n}$ is the outward normal vector to the surface. Finally, this is used to calculate the coefficient of drag,

$$C_d = \frac{F}{\frac{1}{2}\rho u^2 A_p}, \tag{4}$$

where F is the resulting force in flow direction and A_p is the area projected in the direction of flow. Furthermore, the Reynolds number was calculated using the following formula,

$$Re = \frac{v L_c \rho}{\mu}, \tag{5}$$

where $L_c = \sqrt{A_p}$ is the characteristic length. The Reynolds number is calculated since it is a parameter of utmost importance. This gives the data needed to proceed with building the networks.

B. Machine learning.

This part of the theory is split up between the theory behind the neural networks and the theory behind the support vector regression.

1) Support Vector Regression (SVR): SVR is a supervised machine learning method used for predicting unseed numerical values based on a set of known values and feutures first proposed in 1996 by Drucker et al. [11]. The method is similar to Support vector machine (SVM), but focuses on predicting continuous outputs instead of classification. It does this by fitting a n-dimensional hyperplane such that it minimizes the prediction error, in other words find the function $f(x)$ that deviates the least from the target values. In the case of a linear function f can be written as

$$f(x) = \langle \omega, x \rangle + b, \quad \omega \in \mathbb{N}, b \in \mathbb{R}. \tag{6}$$

More formally, SVR can be written as a convex optimization problem [12],

$$\begin{aligned} &\textit{minimize} \quad \frac{1}{2}||\omega||^2, \\ &\textit{subject to} \begin{cases} y_i - \langle \omega, x_i \rangle - b \leq \varepsilon \\ \langle \omega, x_i \rangle + b - y_i \leq \varepsilon \end{cases}, \end{aligned} \tag{7}$$

where ε is the maximum allowed deviation from the target, $\langle \cdot, \cdot \rangle$ denote the dot product, x_i training features and y_i targets such that $(x_i, y_i) \subset \mathbb{R} \times \mathbb{N}$,where $\mathbb{N}$ denotes the space of the input patterns. The problem is not always feasible therefore a slack variable ξ is introduced,

$$\begin{aligned} &\textit{minimize} \quad \frac{1}{2}||\omega||^2 + C \sum (\xi_i + \xi_i^*), \\ &\textit{subject to} \begin{cases} y_i - \langle \omega, x_i \rangle - b \leq \varepsilon + \xi_i \\ \langle \omega, x_i \rangle + b - y_i \leq \varepsilon + \xi_i * \\ \xi_i, \xi_i^* \geq 0 \end{cases}. \end{aligned} \tag{8}$$

In this formulation both C and ε serves as tunable hyperparameter where C determines the trade-off between the flatness of the function and the allowed deviations larger than ε.

2) Neural Networks & Convolutional Neural Networks: A neural network is a structure baised on the brain and how it functions. Each neuron takes a signal from the previous which is the the activation of the sum of all previous signals. This paired with a individual bias for the neuron in turn produces its signal and so on. Mathematically this means the following,

$$y_j = ReLu(\sum_i x_i w_{i,j} + \theta_j), \tag{9}$$

where y_j is the output from neuron j, x_i is the input from neuron i in the previous layer, $w_{i,j}$ is the weight for the connection from i to j and finally θ_j is the bias of neuron j. Here the $ReLu$ function is used, which is a common activation function which is defined as,

$$ReLu(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}. \tag{10}$$

The output from the entire network then needs to be evaluated in order to increase the performance over time. This is done most commonly with the help of MSE loss, mean squared error. This is defined as,

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2, \tag{11}$$

where Y_i is the correct value and $\hat{Y}_i$ is the approximated value. To make the network perform better over time its weights, $w_{i,j}$, and biases, θ_j, need to be updated. This is done with the help of the $pyTorch$ library [13]. The most common optimizer in $pyTorch$is $Adams$ which also is the optimizer used in this report [13]. Different optimizers utelize different algorithms to update the networks parameters, however, these algorithms are mostly variants of gradient descent. To illustrate this process, gradient descent is defined as,

$$a_{n+1} = a_n - \gamma \nabla F(a_n), \tag{12}$$

where a_n and a_{n+1} are points at different iterations, γ is the stepsize or learning rate and ∇F is a gradient used to ensure the correct direction. In machine learning applications F is usually the loss function discussed earlier.

The convolutional neural network works in a different way than the outline described earlier. The convolutional neural network is built on the convolution operation. This allows the network to evaluate images. The convolution is made between the input image and filters. After a fixed number of convolutions the output is flattened, after this point the networks work similarly. The convolution operation between the image and a filter is defined as,

$$(f * g)[i, j] = \sum_k \sum_l g(k, l) \cdot f(i - k, j - l), \tag{13}$$

where f is the image and g is the filter.

III. Method

In this section the methods used and how they were implemeted is presented. This includes everything from generating the datasets used to the implementation of the different models. The section is in turn split up between the data generation and machine learning elements of the methology used.

A. Particle & Data generation.

In order to generate particles an entire new grid, in the form of a matrix, was constructed. This matrix is denoted $A_{i,j,k}$ in algorithm 1. This time each node in the grid represented a possible point where a particle of radius $r = 0.5 \cdot 10^{-3}$ could be placed. As previously stated particles were allowed to overlap with one radius or zero. This meant that throughout the whole grid a node was placed at r length units away from each other. Then a particle was placed in the middle of this grid. After that the following algorithm was used to place the remaining particles.

Algorithm 1 Generate soot particle.

function SOOT($A, Directions, Positions, N_p$)
  while $\sum_{i,j,k} A_{i,j,k} < N_p$ **do**
    $StartPoint = rand(Positions)$
    $Move = rand(N_d)$

    $Change = Directions(Move)$
    $NewIndex = StartPoint + Change$
    $A_{NewIndex} = 1$

    $Positions.append(NewIndex)$
  end while
  return $A, Positions$
end function

Algorithm 1 places particles at random finite locations with respect to a random already placed particle. Later on the new index is added to a list of placed particles if this position. Then simply start over and place a new particle until all $N_p \in \mathbb{N}$ particles have been placed. To filter only the unique positions algorithm 2 is used.

Algorithm 2 Unique particle positions.

function UNIQUE($Position$)
  $UniquePos = []$
  for i in $Positions$ **do**
    $UniqueStatus = 0$
    for j = i+1 in $Positions$ **do**
      if i == j **then**
        $UniqueStatus = 1$
      end if
    end for

    if $UniqueStatus$ == 0 **then**
      $UniquePos.append(i)$
    end if
  end for
  return $UniquePos$
end function

With these algorithms there is sufficient information provided for the simulation software to use to simulate over the created particle and thus creating a dataset. Provided that they are properly integrated before simulating. This integration is providing the particles with material properties for the CFD solver as well as a no slip condition on the surface. The simulation is then done for a channel flow where the particles generated are placed in the middle of the computational domain. The solution is a steady-state solution and no turbulent effects are simulated. Thus appropriate fluid velocities are used.

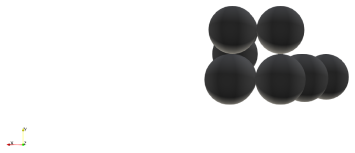

Figure 1: Example particle generated with 15 subspheres. Generated using *pyvista* library for python [14].

Finally, pictures from each particle taken from three different angles were also saved in order to be used for training. These three angles were, from straight ahead, from the side as well as from above (YZ, XZ, XY-planes).

B. Feature selection.

When training or fitting a model proper feature selection is vital. If the features has no or little correlation with the desired output the resulting model would perform poorly. To select good features for model training a correlation heatmap was created.

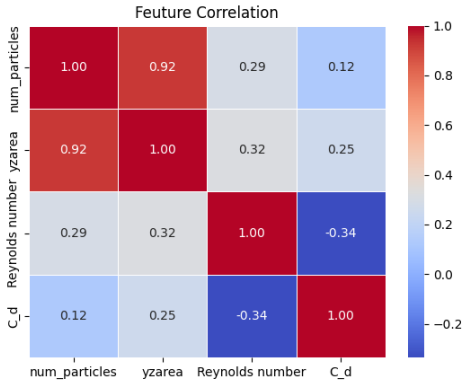

Figure 2: Correlation heatmap of features, made using *seaborn* [15].

In figure 2 one can see the Pearson-correlations between the features. This is calculated by the following,

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}, \tag{14}$$

where σ denotes the standard deviation of X and Y respectivly, which in turn are two random variables representing different quantities.

It is clear and somewhat expected that the highest correlation with C_d is with the reynolds number and the projected area in the direction fo the flow. These features are clear candidates for training a machine learning model, but since the numbers of features used when training is not limited, features with lower correlation can still be used and provide the model with additional information.

C. Machine learning.

In this subsection the method revolving around the machine learning techniques used will be explained and discussed. This involves support vector regression as well as stacked networks. For all methods 80% of the data set was used for training the models while the other 20% was used for testing the models. The training data was in turn split in to 90% training- and 10% validation data

1) Support Vector Regression: As a baseline a Support Vector regression (SVR) model was as metioned previously implemeted to serve as a comparison to the proposed stacked network. This was done since SVR offers a very quick and simple setup, yet good performance. This model was trained on reynolds number and projected area, chosen after analyzing the corellation between features and C_d. This method was implemeted using the python *scikit-learn* library and the tools they provide [16].

After feature selection the features are scaled to ensure that the model interperet the features on an equal scale, otherwise features with larger derivations might have a larger impact on the model. For this study the *StandardScaler* is used to scale the features. This scaler removes the mean and scales to unit variance.

After the features are scaled, kernels and hyperparameters are selectet. The RBF-kernel was chosen and since SVR only serves as a baseline comparison the default hyperparameters was chosen. If one where to optimize SVR a grid search of hyperparameters neeeds to be performed. Finally, before evaluating the performance all data has to be inverse-transformed using the same scaler used when scaling them before fitting the model to the dataset.

2) Ensamble Neural Network: For the deep neural network, it was implemented as seen in figure 3.

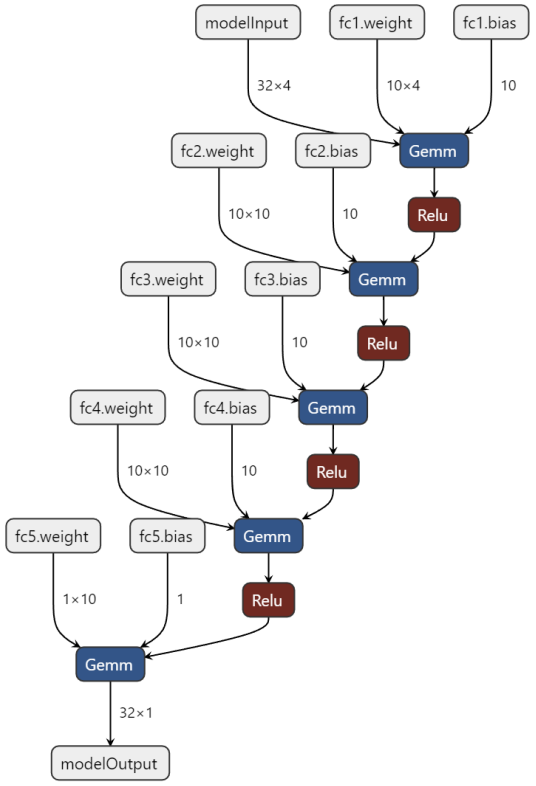

Figure 3: Structure of neural network, made using Netron [17].

Algorithm 3 Train Deep Neural Network

1: **function** TRAIN($DNN, Data_T, Data_V, n_B, n_E$)
2: **for** i in $range(n_E)$ **do**
3: **if** $i = 5$ **then** l_r -= 0.01
4: **end if**
5: **for** j in $Data_T$ **do**
6: $Loss = DNN(Data_T(i : i + n_B))$
7: $optimize(DNN, loss)$
8: **end for**
9: **for** i in $Data_V$ **do**
10: $Loss_V = DNN(Data_V(i : i + n_B))$
11: **if** $Loss_V > Loss_V(i - 1)$ **then**
12: **return** DNN
13: **end if**
14: **if** $Loss_V \leq tol$ **then return** DNN
15: **end if**
16: **end for**
17: **end for**
18: **end function**

This network was then trained in a standard manner with data previously generated. This network also utelized early stopping in order to not overfit on the training data. Another precaution made to avoid overfitting was the fact that validation data was used to calculate the loss for each epoch.

In algorithm 3, DNN denotes the deep neural network model, $Data_T$ denotes the training data, $Data_V$ denotes the validation data. Furthermore, n_B denotes the batchsize, n_E denotes the maximum number of iterations to perform. The algorith in short trains the model. It goes through the training data and optimizes the network with respect to the loss, then it goes through the validation data and if the model performes especially poorly or especially well the training is suspended. This is the early stopping used to avoid overfitting the model to the training data. Furthermore, another key feature used in order to increase overall performance is the learning rate scheduling which can be seen on lines $3 - 4$. The learning rate is lowered after 5 iterations which, in theory, should increase training performance of the algorithm.

3) Stacked Networks: The deep neural network was tasked with predicting the coefficent of drag, C_d, from the number of spheres in the particle, the Reynolds number, the projected area of the particle in the direction of the flow and the radius of the spheres. The structure of the network is seen in figure 3.

The convolutional neural network was used to correct for the error of the neural network. This meaning, with the help of 3 pictures of the particle taken in the YZ-plane,XZ-plane and XY-plane the network predicted the error that the neural network produced. The training of the convolutional network closely resembles the same as for the deep neural network in algorithm 3, therefore it is not shown explicitly.

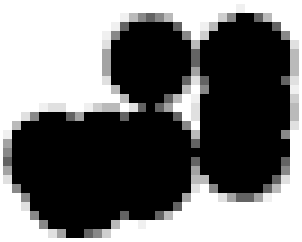

Figure 4: Picture used for training CNN taken from XY-direction.

The structure of this network is seen below in figure 5.

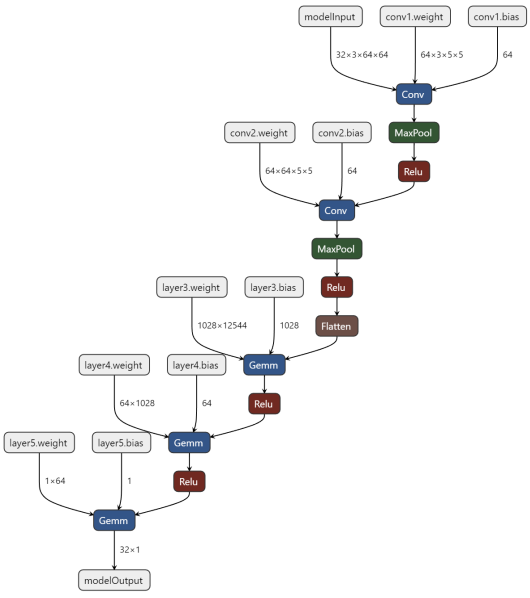

Figure 5: Structure of CNN, made with Netron [17].

All of this was then fed to the final neural network which predicted the coefficent of drag, C_d, using the initial prediction from the neural network as well as its predicted error from the convolutional neural network. The overall structure is made clearer in figure 6.

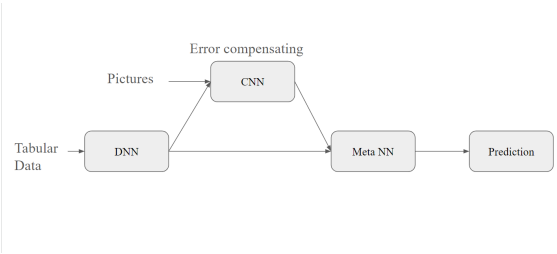

Figure 6: Model overview.

All of these networks were trained individually and seperately. This allows for closer inspection of each network as well as individual tweaking of each network in order to optimize the overall performance.

IV. Results and Discussion

In this section of the paper the results will be shown for the different approaches used. First off is the results from the support vector regression endevour and later on the results from the neural network as well as the stacked network. Furthermore, the results will be discussed.

A. *Support Vector Regression.*

The SVR models prediction compared to the correct values can be seen in the figure below 7.

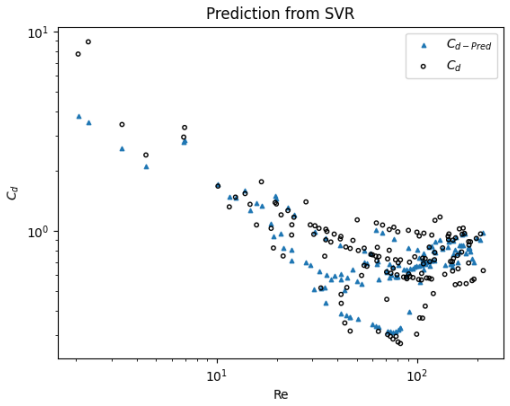

Figure 7: SVR predictions.

One can see in the figure that SVR performes quite well considering the only features used was reynolds number and projected area. It manages to catch the main trend but also many outliers that deviates. The

resulting metrics was a RMSE of ≈ 0.22, a MAPE of ≈ 0.23 and a r^2-score of ≈ 0.77.

Interestingly SVR did not require that many datapoints for training to reach the same performance as when 80% of the dataset was used which can be seen in the figure below 8

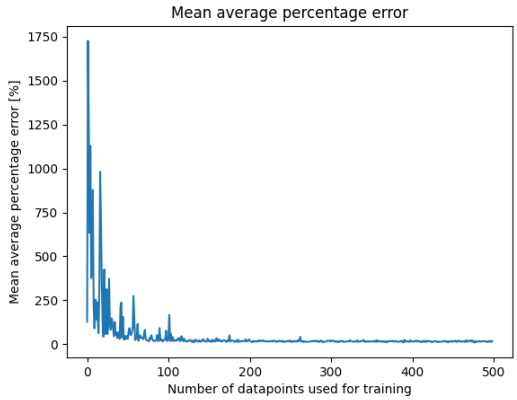

Figure 8: Relations between performance and number of training points.

B. *Deep neural network.*

The predictions that the deep neural network made is shown in figure 9. It is clear that the deep neural network finds the general shape of C_d. However, for the larger Reynolds numbers it can not take geometrical effects in to account which is clear since the graph shows the predictions to not follow the data closely in this region.

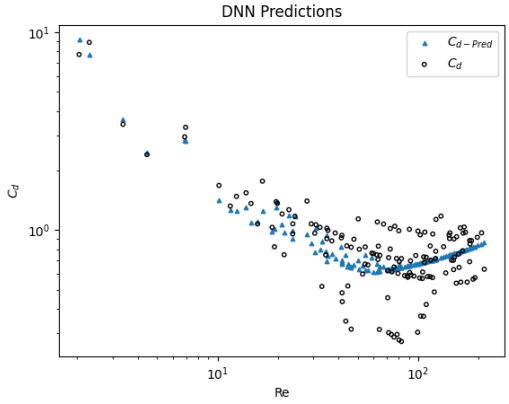

Figure 9: Predictions made by the deep neural network.

In figure 10 the convergence for the deep neural network is shown. It converges quickly and reaches a reletively low point. The reason it is stopped is due to the fact that early stopping was implemented. This means that the loss is not allowed to increase, the training would then be suspended.

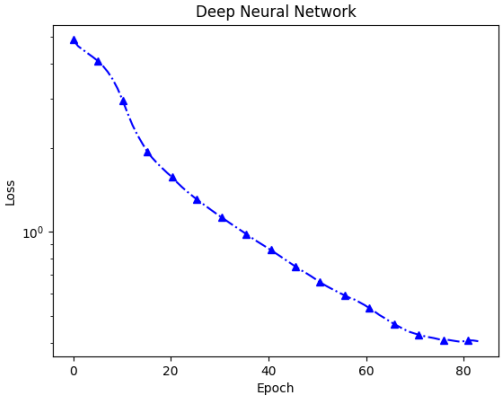

Figure 10: Convergence of the deep neural network.

Furthermore, the deep neural network managed to get a RMSE of ≈ 0.28, a MAPE of ≈ 0.27 and finally a r^2-score of ≈ 0.92. This is deemed as decent performance, however, it is worse than the baseline set by the SVR.

C. *Stacked Network.*

The result is shown in figure 11. In the left-most plot the deep neural nets predictions as well as the test data are shown in blue and black respectively. In the middle plot the error from deep neural network is shown in black as the target data as well as the predictions made from the convolutional neural network shown in blue. Finally, the right-most plot shows once again the test data in black and the predictions made by, in this case, the stacked network in blue.

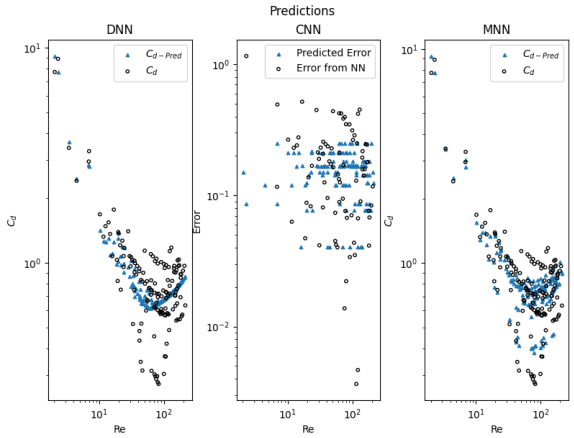

Figure 11: Predictions made by the model and individual networks.

The stacked network managed to get an $r^2 - score \approx 0.96$, $RMSE$, of ≈ 0.22 and a $MAPE$, of ≈ 0.16. This result is good. It also shows a clear cooperation between the ingoing networks, since these networks performe worse individually. Looking at the convergence, in figure 12, there is clear support for the idea of cooperation between the networks as the stacked network outperformes both other networks individually.

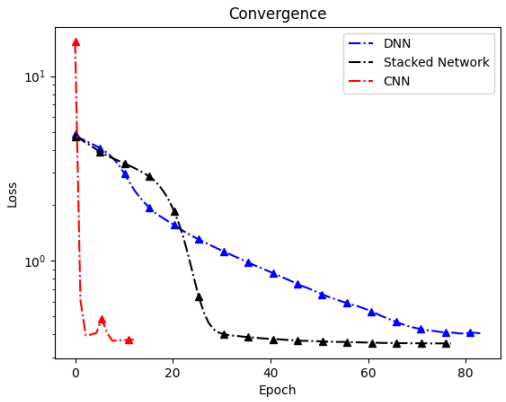

Figure 12: Convergence for the different networks.

It is clear that the stacked network outperformces even the deep neural network individually.

D. Comparison between models.

When comparing the models three common metrics where used, root mean squared error (RMSE), mean average percentage error (MAPE) and coefficient of determination (r^2-score). These provide information aboute the absoulte error, percetage error and how well the model explains the variations in the data.

As seen in table I, according to these metrics the stacked neural network performs significantly better than SVR and DNN.

Table I: Comparison of stastistical data from models.

Model/Metric	SVR	DNN	Stacked
RMSE	0.22	0.28	0.22
MAPE	0.23	0.27	0.16
r^2-score	0.77	0.92	0.96

V. Conclusion

In summary, three different methods were examnined and compared. They were all evaluated on their ability to reliably predict C_d. In order to do this psuedo-random particles were generated and then simulated over. This data was then used to train the different models. Finally their performance was compared using different statistics such as RMSE and MAPE.

In conclusion the stacked network model works well. It is able to predict the coefficient of drag C_d for different structures of the particles. It is also clear that the stacking of the networks has a benefitial affect on the overall performance. This is clear since the deep neural network is not able to predict changes in the coefficient of drag C_d due mainly to changes in geometry, which can be seen in figure 11 in the left-most graph. In the higher Reynolds number region the spread of the data is greater, which suggests that the geometry has a larger impact on the final value of the drag coefficient. The convolutional neural network corrects for this which is clear from the right most graph in the same figure. Overall the performance of the stacked network model performces the best even compared to the SVR model baseline.

VI. Further developement

The authors of this study recognizes the limitations of this study and several ways upon which it can be improved. In this sections a few ideas are presented that the readers are encouraged to use to further improve the work of this study.

These ideas include using a pretrained convolutional nerual network, such as ResNet50, in order to increase its performance despite the relatively low number of data points available, which is an overall

limiting factor. Another idea would be to investigate different and more combinations of models, not only the combination of a deep neural network with a convolutional neural network, but also to incorporate the use of SVR in this and all of the possible combinations of models. Further work might consist of predicting wall effects. This meaning, trying to predict what the coefficient of drag will be when the particle is particularly close to a wall. Other interesting areas would be to introduce turbulence and or trying to implement physical laws or empirical correlations as [1] has shown to be promising.

ACKNOWLEDGMENT

We want to thank Andreas Mark and Thomas Johnson for the opportunity to work with this project and for the guidance and support along the way. As well as the rest of our collegues at *FCC* for the great support.

References

[1] M. Presa-Reyes, P. Mahyawansi, B. Hu, D. McDaniel, and S.-C. Chen, "Dcc-dnn: A deep neural network model to predict the drag coefficients of spherical and non-spherical particles aided by empirical correlations," *Powder Technology*, vol. 435, p. 119388, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0032591024000305

[2] S. K. Sanjeevi, J. Kuipers, and J. T. Padding, "Drag, lift and torque correlations for non-spherical particles from stokes limit to high reynolds numbers," *International Journal of Multiphase Flow*, vol. 106, pp. 325–337, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301932217307851

[3] E. Loth, "Drag of non-spherical solid particles of regular and irregular shape," *Powder Technology*, vol. 182, no. 3, pp. 342–353, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0032591007002938

[4] A. Hölzer and M. Sommerfeld, "Lattice boltzmann simulations to determine drag, lift and torque acting on non-spherical particles," *Computers Fluids*, vol. 38, no. 3, pp. 572–589, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045793008001175

[5] C. Xie, T. Song, and Y. Zhao, "Discrete element modeling and simulation of non-spherical particles using polyhedrons and super-ellipsoids," *Powder Technology*, vol. 368, pp. 253–267, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0032591020303697

[6] M. Zastawny, G. Mallouppas, F. Zhao, and B. van Wachem, "Derivation of drag and lift force and torque coefficients for non-spherical particles in flows," *International Journal of Multiphase Flow*, vol. 39, pp. 227–239, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301932211002047

[7] Y. Chen and C. Müller, "Development of a drag force correlation for assemblies of cubic particles: The effect of solid volume fraction and reynolds number," *Chemical Engineering Science*, vol. 192, pp. 1157–1166, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0009250918305967

[8] M. Mandø and L. Rosendahl, "On the motion of non-spherical particles at high reynolds number," *Powder Technology*, vol. 202, no. 1, pp. 1–13, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0032591010002391

[9] M. Nikku, P. Jalali, J. Ritvanen, and T. Hyppänen, "Characterization method of average gas–solid drag for regular and irregular particle groups," *Powder Technology*, vol. 253, p. 284–294, 02 2014.

[10] F. Batlouni, B. Elm Jonsson, O. Fjeldså, N. Persson, and L. Ånestrand, "Utveckling av turbulensmodeller med hjälp av maskininlärning i python," 2023. [Online]. Available: http://hdl.handle.net/20.500.12380/306750

[11] H. Drucker, C. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Linear support vector regression machines," *Advances in neural information processing systems*, vol. 9, no. 9, pp. 155–161, 1996.

[12] D. Basak, S. Pal, and D. Patranabis, "Support vector regression," *Neural Information Processing – Letters and Reviews*, vol. 11, 11 2007. [Online]. Available: https://www.researchgate.net/publication/228537532_Support_Vector_Regression

[13] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, M. Suo, P. Tillet, E. Wang, X. Wang, W. Wen, S. Zhang, X. Zhao, K. Zhou, R. Zou, A. Mathews, G. Chanan, P. Wu, and S. Chintala, "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. [Online]. Available: https://pytorch.org/assets/pytorch2-2.pdf

[14] C. B. Sullivan and A. Kaszynski, "PyVista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (VTK)," *Journal of Open Source Software*, vol. 4, no. 37, p. 1450, may 2019. [Online]. Available: https://doi.org/10.21105/joss.01450

[15] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. [Online]. Available: https://doi.org/10.21105/joss.03021

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[17] "Netron: Visualizer for neural network, deep learning and machine learning models." [Online]. Available: https://github.com/lutzroeder/netron