Appendix to "Automated Assessment of Residual Plots with Computer Vision Models"

Weihao Li[a,b], Dianne Cook[a], Emi Tanaka[a,b,c], Susan VanderPlas[d], Klaus Ackermann[a]

[a]Department of Econometrics and Business Statistics, Monash University, Clayton, VIC, Australia; [b]Biological Data Science Institute, Australian National University, Acton, ACT, Australia; [c]Research School of Finance, Actuarial Studies and Statistics, Australian National University, Acton, ACT, Australia; [d]Department of Statistics, University of Nebraska, Lincoln, Nebraska, USA

ARTICLE HISTORY

Compiled June 17, 2025

Appendix A. Data Generation

A.1. *Simulation Scheme*

While observational data is frequently employed in training models for real-world applications, the data generating process of observational data often remains unknown, making computation for our target variable D unattainable. Consequently, the computer vision models developed in this study were trained using synthetic data, including 80,000 training images and 8,000 test images. This approach provided us with precise label annotations. Additionally, it ensured a large and diverse training dataset, as we had control over the data generating process, and the simulation of the training data was relatively cost-effective.

We have incorporated three types of residual departures of linear regression model in the training data, including non-linearity, heteroskedasticity and non-normality. All three departures can be summarized by the data generating process formulated as

Table A1. Factors used in the data generating process for synthetic data simulation. Factor j and a controls the non-linearity shape and the heteroskedasticity shape respectively. Factor b, σ_ε and n control the signal strength. Factor dist_ε, dist_{x1} and dist_{x2} specifies the distribution of ε, X_1 and X_2 respectively.

Factor	Domain
j	{2, 3, ..., 18}
a	[-1, 1]
b	[0, 100]
β_1	{0, 1}
β_2	{0, 1}
dist_ε	{discrete, uniform, normal, lognormal}
dist_{x1}	{discrete, uniform, normal, lognormal}
dist_{x2}	{discrete, uniform, normal, lognormal}
σ_ε	[0.0625, 9]
σ_{X1}	[0.3, 0.6]
σ_{X2}	[0.3, 0.6]
n	[50, 500]

$$\boldsymbol{y} = \boldsymbol{1}_n + \boldsymbol{x}_1 + \beta_1\boldsymbol{x}_2 + \beta_2(\boldsymbol{z} + \beta_1\boldsymbol{w}) + \boldsymbol{k} \odot \boldsymbol{\varepsilon}, \tag{A1}$$

$$\boldsymbol{z} = \text{He}_j(g(\boldsymbol{x}_1, 2)), \tag{A2}$$

$$\boldsymbol{w} = \text{He}_j(g(\boldsymbol{x}_2, 2)), \tag{A3}$$

$$\boldsymbol{k} = \left[\boldsymbol{1}_n + b(2 - |a|)(\boldsymbol{x}_1 + \beta_1\boldsymbol{x}_2 - a\boldsymbol{1}_n)^{\circ 2}\right]^{\circ 1/2}, \tag{A4}$$

where $\boldsymbol{y}$, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{z}$, $\boldsymbol{w}$, $\boldsymbol{k}$ and $\boldsymbol{\varepsilon}$ are vectors of size n, $\boldsymbol{1}_n$ is a vector of ones of size n, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are two independent predictors, $\text{He}_j(.)$ is the jth-order probabilist's Hermite polynomials (Hermite 1864), $(.)^{\circ 2}$ and $(.)^{\circ 1/2}$ are Hadamard square and square root, $\odot$ is the Hadamard product, and $g(\boldsymbol{x}, k)$ is a scaling function to enforce the support of the random vector to be $[-k, k]^n$ defined as

$$g(\boldsymbol{x}, k) = 2k \cdot \frac{\boldsymbol{x} - x_{\min}\boldsymbol{1}_n}{x_{\max} - x_{\min}} - k\boldsymbol{1}_n,\ for\ k > 0,$$

where $x_{\min} = \min\limits_{i\in\{1,\ldots,n\}} x_i$, $x_{\max} = \max\limits_{i\in\{1,\ldots,n\}} x_i$ and x_i is the i-th entry of $\boldsymbol{x}$.

The residuals and fitted values of the fitted model were obtained by regressing $\boldsymbol{y}$

on $\boldsymbol{x}_1$. If $\beta_1 \neq 0$, $\boldsymbol{x}_2$ was also included in the design matrix. This data generation process was adapted from Li et al. (2024), where it was utilized to simulate residual plots exhibiting non-linearity and heteroskedasticity visual patterns for human subject experiments. A summary of the factors utilized in Equation A1 is provided in Table A1.

In Equation A1, $\boldsymbol{z}$ and $\boldsymbol{w}$ represent higher-order terms of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, respectively. If $\beta_2 \neq 0$, the regression model will encounter non-linearity issues. Parameter j serves as a shape parameter that controls the number of tuning points in the non-linear pattern. Typically, higher values of j lead to an increase in the number of tuning points, as illustrated in Figure A1.

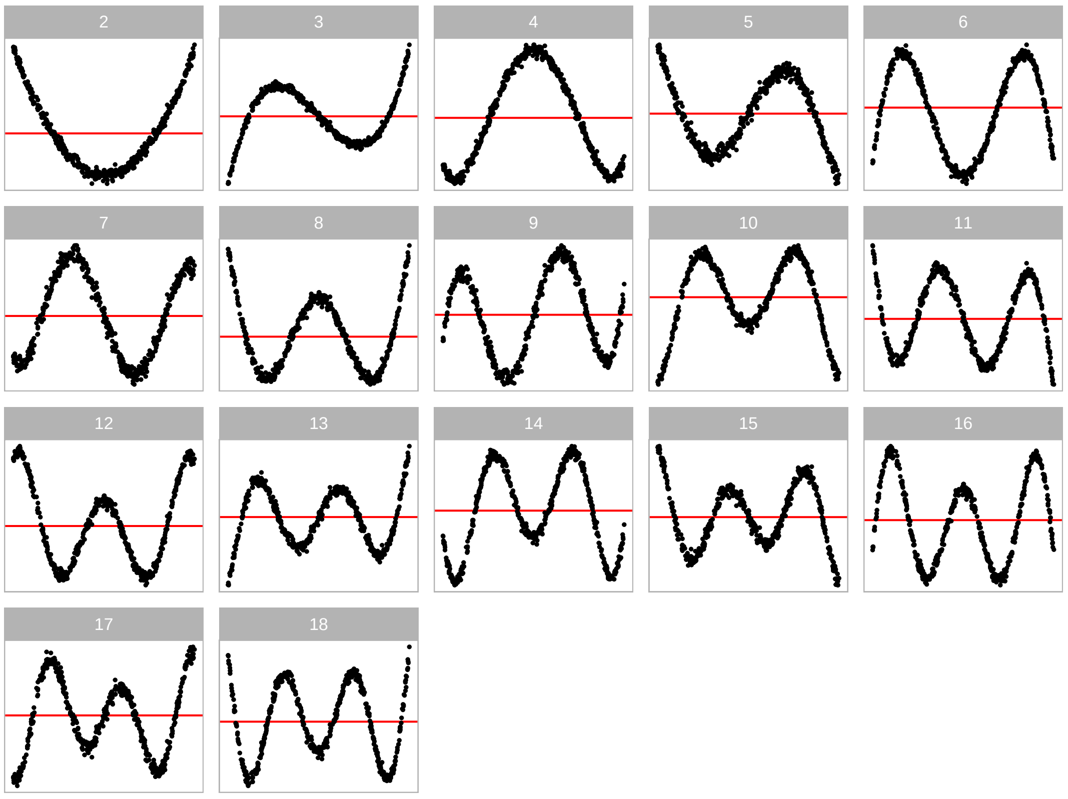

Figure A1. Non-linearity forms generated for the synthetic data simulation. The 17 shapes are generated by varying the order of polynomial given by j in $He_j(.)$.

Additionally, scaling factor $\boldsymbol{k}$ directly affects the error distribution and it is correlated with $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. If $b \neq 0$ and $\boldsymbol{\varepsilon} \sim N(\boldsymbol{0}_n, \sigma^2 \boldsymbol{I}_n)$, the constant variance assumption will be violated. Parameter a is a shape parameter controlling the location of the smallest variance in a residual plot as shown in Figure A2.

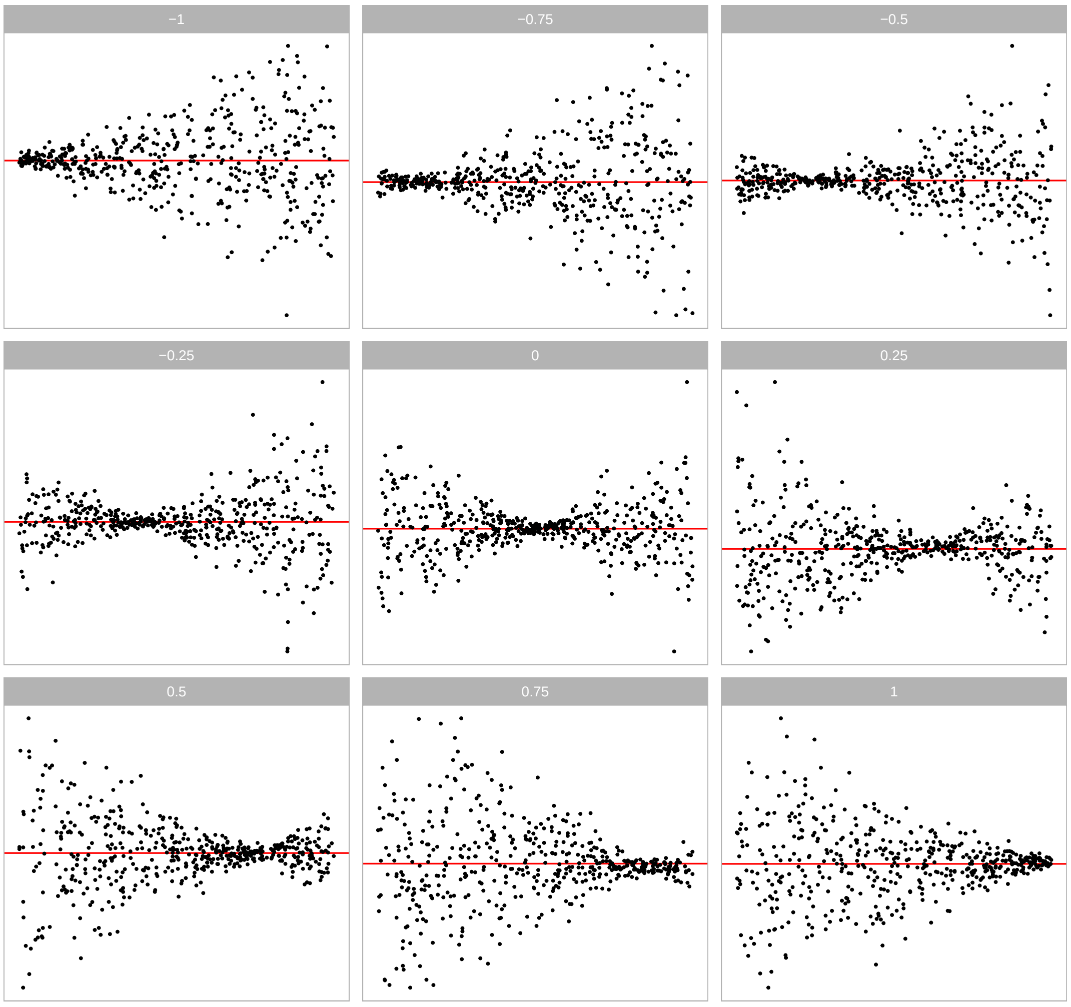

Figure A2. Heteroskedasticity forms generated for the synthetic data simulation. Different shapes are controlled by the continuous factor a between -1 and 1. For $a = -1$, the residual plot exhibits a "left-triangle" shape. And for $a = 1$, the residual plot exhibits a "right-triangle" shape.

Non-normality violations arise from specifying a non-normal distribution for $\boldsymbol{\varepsilon}$. In the synthetic data simulation, four distinct error distributions are considered, including discrete, uniform, normal, and lognormal distributions, as presented in Figure A3. Each distribution imparts unique characteristics in the residual plot. The discrete error distribution introduces discrete clusters in residuals, while the lognormal distribution typically yields outliers. Uniform error distribution may result in residuals filling the entire space of the residual plot. All of these distributions exhibit visual distinctions from the normal error distribution.

Equation A1 accommodates the incorporation of the second predictor $\boldsymbol{x}_2$. Introducing it into the data generation process by setting $\beta_1 = 1$ significantly enhances the

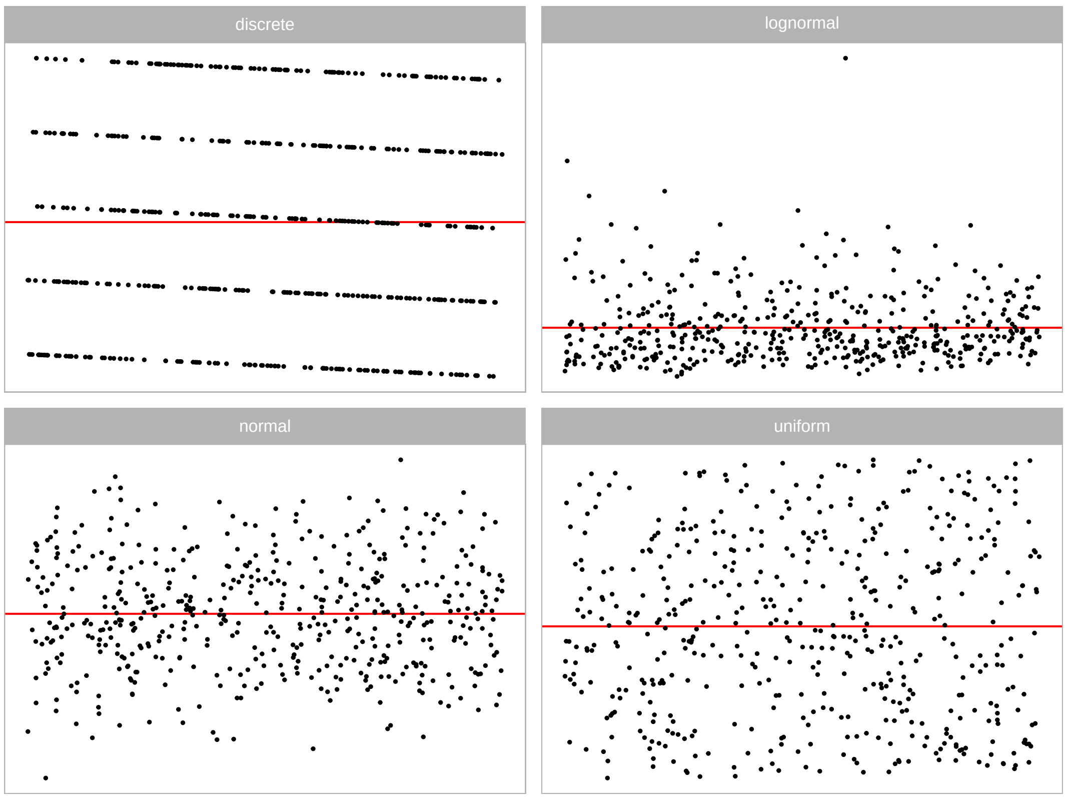

Figure A3. Non-normality forms generated for the synthetic data simulation. Four different error distributions including discrete, lognormal, normal and uniform are considered.

complexity of the shapes, as illustrated in Figure A4. In comparison to Figure A1, Figure A4 demonstrates that the non-linear shape resembles a surface rather than a single curve. This augmentation can facilitate the computer vision model in learning visual patterns from residual plots of the multiple linear regression model.

In real-world analysis, it is not uncommon to encounter instances where multiple model violations coexist. In such cases, the residual plots often exhibit a mixed pattern of visual anomalies corresponding to different types of model violations. Figure A5 and Figure A6 show the visual patterns of models with multiple model violations.

The predictors, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, are randomly generated from four distinct distributions, including $U(-1,1)$ (uniform), $N(0,0.3^2)$ (normal), lognormal$(0,0.6^2)/3$ (skewed) and $U\{-1,1\}$ (discrete uniform).

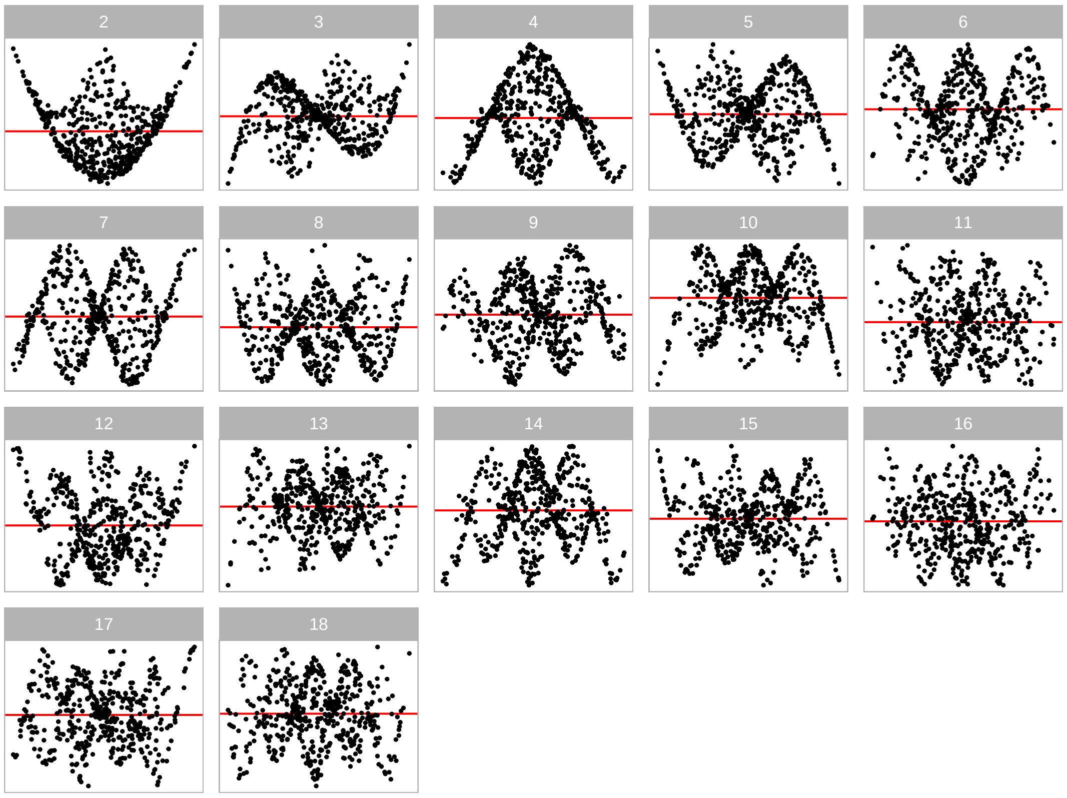

Figure A4. Residual plots of multiple linear regression models with non-linearity issues. The 17 shapes are generated by varying the order of polynomial given by j in $He_j(.)$. A second predictor $\boldsymbol{x}_2$ is introduced to the regression model to create complex shapes.

A.2. *Balanced Dataset*

To train a robust computer vision model, we deliberately controlled the distribution of the target variable D in the training data. We ensured that it followed a uniform distribution between 0 and 7. This was achieved by organizing 50 buckets, each exclusively accepting training samples with D falling within the range $[7(i-1)/49, 7i/49)$ for $i < 50$, where i represents the index of the i-th bucket. For the 50-th bucket, any training samples with $D \geq 7$ were accepted.

With 80,000 training images prepared, each bucket accommodated a maximum of $80000/50 = 1600$ training samples. The simulator iteratively sampled parameter values from the parameter space, generated residuals and fitted values using the data generation process, computed the distance, and checked if the sample fitted within the corresponding bucket. This process continued until all buckets were filled.

Similarly, we adopted the same methodology to prepare 8,000 test images for per-

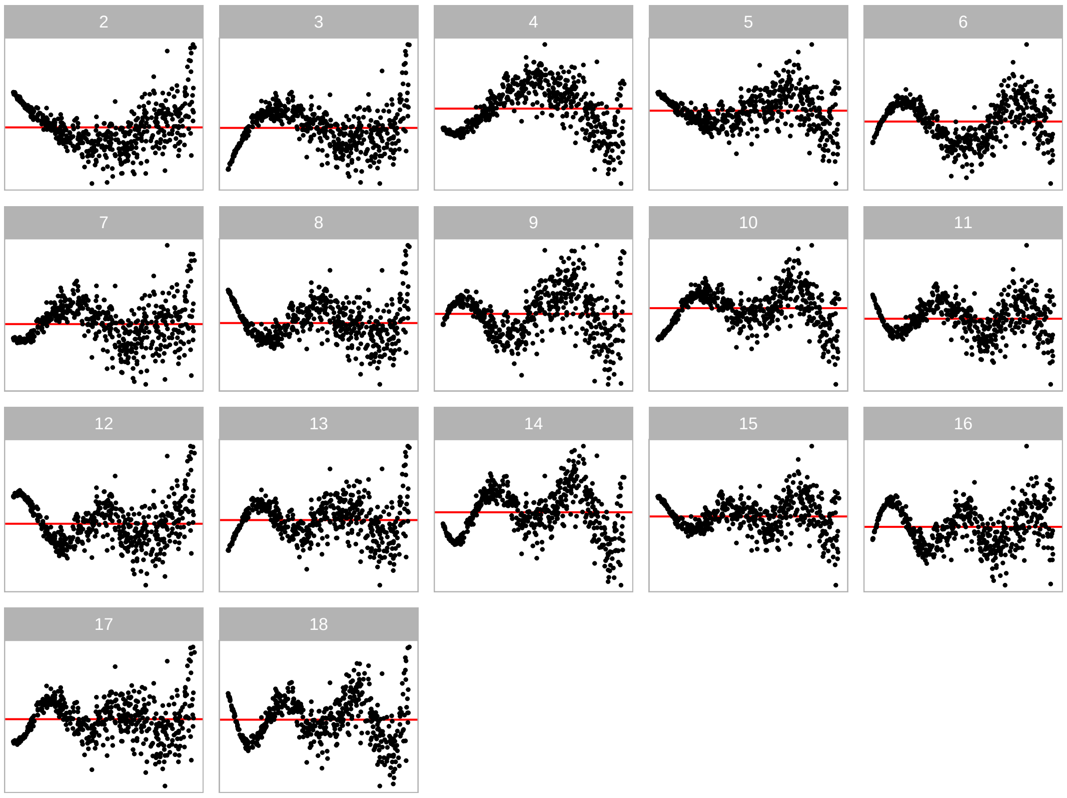

Figure A5. Residual plots of models violating both the non-linearity and the heteroskedasticity assumptions. The 17 shapes are generated by varying the order of polynomial given by j in $He_j(.)$, and the "left-triangle" shape is introduced by setting $a = -1$.

formance evaluation and model diagnostics.

A.3. *Neural Network Layers Used in the Study*

This study used seven types of neural network layers, all of which are standard components frequently found in modern deep learning models. These layers are well-documented in textbooks like Goodfellow, Bengio, and Courville (2016) and Chollet (2021), which offer thorough explanations and mathematical insights. In this section, we will offer a concise overview of these layers, drawing primarily from the insights provided in Goodfellow, Bengio, and Courville (2016).

A.3.1. Dense Layer

The Dense layer, also known as the fully-connected layer, is the fundamental unit in neural networks. It conducts a matrix multiplication operation between the input

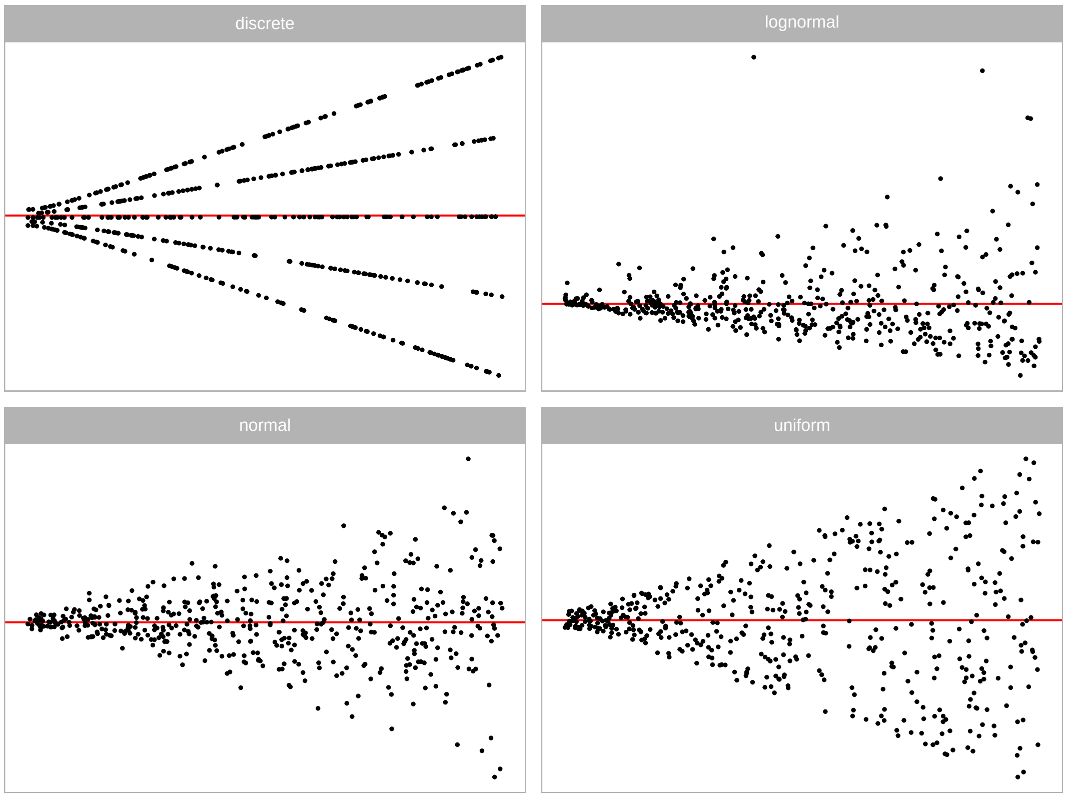

Figure A6. Residual plots of models violating both the non-normality and the heteroskedasticity assumptions. The four shapes are generated by using four different error distributions including discrete, lognormal, normal and uniform, and the "left-triangle" shape is introduced by setting $a = -1$.

matrix $\boldsymbol{I}$ and a weight matrix $\boldsymbol{W}$ to generate the output matrix $\boldsymbol{O}$, which can be written as

$$\boldsymbol{O} = \boldsymbol{I}\boldsymbol{W} + b,$$

where b is the intercept.

A.3.2. ReLu Layer

The ReLU layer, short for rectified linear unit, is an element-wise non-linear function introduced by Nair and Hinton (2010). It sets the output elements to zero if the corresponding input element is negative; otherwise, it retains the original input. Mathematically, it can be expressed as:

$$\boldsymbol{O}(i,j) = max\{0, \boldsymbol{I}(i,j)\},$$

where $\boldsymbol{O}(i,j)$ is the ith row and jth column entry of matrix $\boldsymbol{O}$, and $\boldsymbol{I}(i,j)$ is the ith row and jth column entry of matrix $\boldsymbol{I}$.

A.3.3. Convolutional Layer

In Dense layers, matrix multiplication leads to each output unit interacting with every input unit, whereas convolutional layers operate differently with sparse interactions. An output unit in a convolutional layer is connected solely to a subset of input units, and the weight is shared across all input units. Achieving this involves using a kernel, typically a small square matrix, to conduct matrix multiplication across all input units. Precisely, this concept can be formulated as:

$$\boldsymbol{O}(i,j) = \sum_m \sum_n \boldsymbol{I}(i-m, j-n)K(m,n),$$

where m and n are the row and columns indices of the kernel K.

If there are multiple kernels used in one covolutional layer, then each kernel will have its own weights and the output will be a three-dimensional tensor, where the length of the third channel is the number of kernels.

A.3.4. Pooling Layer

A pooling layer substitutes the input at a specific location with a summary statistic derived from nearby input units. Typically, there are two types of pooling layers: max pooling and average pooling. Max pooling computes the maximum value within a rectangular neighborhood, while average pooling calculates their average. Pooling layers helps making the representation approximately invariant to minor translations of the input. The output matrix of a pooling layer is approximately s times smaller than the input matrix, where s represents the length of the rectangular area. A max pooling layer can be formulated as:

$$\boldsymbol{O}(i,j) = \max_{m,n} \boldsymbol{I}(si+m, sj+n).$$

A.3.5. Global Pooling Layer

A global pooling layer condenses an input matrix into a scalar value by either extracting the maximum or computing the average across all elements. This layer acts as a crucial link between the convolutional structure and the subsequent dense layers in a neural network architecture. When convolutional layers uses multiple kernels, the output becomes a three-dimensional tensor with numerous channels. In this scenario, the global pooling layer treats each channel individually, much like distinct features in a conventional classifier. This approach facilitates the extraction of essential features from complex data representations, enhancing the network's ability to learn meaningful patterns. A global max pooling layer can be formulated as

$$O(i,j) = \max_{m,n,k} I(si+m, sj+n, k),$$

where k is the kernel index.

A.3.6. Batch Normalization Layer

Batch normalization is a method of adaptive reparametrization. One of the issues it adjusts is the simultaneous update of parameters in different layers, especially for network with a large number layers. At training time, the batch normalization layer normalizes the input matrix I using the formula

$$\boldsymbol{O} = \frac{\boldsymbol{I} - \boldsymbol{\mu}_I}{\boldsymbol{\sigma}_I},$$

where $\boldsymbol{\mu}_I$ and $\boldsymbol{\sigma}_I$ are the mean and the standard deviation of each unit respectively.

It reparametrizes the model to make some units always be standardized by definition, such that the model training is stabilized. At inference time, $\boldsymbol{\mu}_I$ and $\boldsymbol{\sigma}_I$ are usually replaced with the running mean and running average obtained during training.

A.3.7. Dropout Layer

Dropout is a computationally inexpensive way to apply regularization on neural network. For each input unit, it randomly sets to be zero during training, effectively training a large number of subnetworks simultaneously, but these subnetworks share weights and each will only be trained for a single steps in a large network. It is essentially a different implementation of the bagging algorithm. Mathematically, it is formulated as

$$\boldsymbol{O}(i,j) = \boldsymbol{D}(i,j)\boldsymbol{I}(i,j),$$

where $\boldsymbol{D}(i,j) \sim B(1,p)$ and p is a hyperparameter that can be tuned.

Appendix B. Model Training

To achieve a near-optimal deep learning model, hyperparameters like the learning rate often need to be fine-tuned using a tuner. In our study, we utilized the Bayesian optimization tuner from the `KerasTuner` Python library (O'Malley et al. 2019) for this purpose. A comprehensive list of hyperparameters is provided in Table B1.

The number of base filters determines the number of filters for the first 2D convolutional layer. In the VGG16 architecture, the number of filters for the 2D convolutional layer in a block is typically twice the number in the previous block, except for the last block, which maintains the same number of convolution filters as the previous one. This hyperparameter aids in controlling the complexity of the computer vision model. A higher number of base filters results in more trainable parameters. Likewise, the number of units for the fully-connected layer determines the complexity of the final prediction block. Increasing the number of units enhances model complexity, resulting in more trainable parameters.

The dropout rate and batch normalization are flexible hyperparameters that work in conjunction with other parameters to facilitate smooth training. A higher dropout rate is necessary when the model tends to overfit the training dataset by learning too much noise (Srivastava et al. 2014). Conversely, a lower dropout rate is preferred when the model is complex and challenging to converge. Batch normalization, on the other hand, addresses the internal covariate shift problem arising from the randomness in weight initialization and input data (Goodfellow, Bengio, and Courville 2016). It helps stabilize and accelerate the training process by normalizing the activations of each layer.

Additionally, incorporating additional inputs such as scagnostics and the number of observations can potentially enhance prediction accuracy. Therefore, we allow the tuner to determine whether these inputs were necessary for optimal model performance.

The learning rate is a crucial hyperparameter, as it dictates the step size of the optimization algorithm. A high learning rate can help the model avoid local minima but risks overshooting and missing the global minimum. Conversely, a low learning rate smoothens the training process but makes the convergence time longer and increases the likelihood of getting trapped in local minima.

Our model was trained on the MASSIVE M3 high-performance computing platform (Goscinski et al. 2014), using TensorFlow (Abadi et al. 2016) and Keras (Chollet et al. 2015). During training, 80% of the training data was utilized for actual training, while the remaining 20% was used as validation data. The Bayesian optimization tuner conducted 100 trials to identify the best hyperparameter values based on validation root mean square error. The tuner then restored the best epoch of the best model from the trials. Additionally, we applied early stopping, terminating the training process if the validation root mean square error fails to improve for 50 epochs. The maximum allowed epochs was set at 2,000, although no models reached this threshold.

Based on the tuning process described above, the optimized hyperparameter values are presented in Table B2. It was observable that a minimum of 32 base filters was necessary, with the preferable choice being 64 base filters for both the 64×64 and $128 \times$

Table B1. Name of hyperparameters and their correspoding domain for the computer vision model.

Hyperparameter	Domain
Number of base filters	{4, 8, 16, 32, 64}
Dropout rate for convolutional blocks	[0.1, 0.6]
Batch normalization for convolutional blocks	{false, true}
Type of global pooling	{max, average}
Ignore additional inputs	{false, true}
Number of units for the fully-connected layer	{128, 256, 512, 1024, 2048}
Batch normalization for the fully-connected layer	{false, true}
Dropout rate for the fully-connected layer	[0.1, 0.6]
Learning rate	$[10^{-8}, 10^{-1}]$

Table B2. Hyperparameters values for the optimized computer vision models with different input sizes.

Hyperparameter	32×32	64×64	128×128
Number of base filters	32	64	64
Dropout rate for convolutional blocks	0.4	0.3	0.4
Batch normalization for convolutional blocks	true	true	true
Type of global pooling	max	average	average
Ignore additional inputs	false	false	false
Number of units for the fully-connected layer	256	256	256
Batch normalization for the fully-connected layer	false	true	true
Dropout rate for the fully-connected layer	0.2	0.4	0.1
Learning rate	0.0003	0.0006	0.0052

128 models, mirroring the original VGG16 architecture. The optimized dropout rate for convolutional blocks hovered around 0.4, and incorporating batch normalization for convolutional blocks proved beneficial for performance.

All optimized models chose to retain the additional inputs, contributing to the reduction of validation error. The number of units required for the fully-connected layer was 256, a relatively modest number compared to the VGG16 classifier, suggesting that the problem at hand was less complex. The optimized learning rates were higher for models with higher resolution input, likely because models with more parameters are more prone to getting stuck in local minima, requiring a higher learning rate.

Appendix C. Model Violations Index

In Section 5.1, we noted that a pre-computed lattice of $\hat{D}$ quantiles can reduce the computation cost of lineup tests. Another practical approach is to assess model performance directly using the value of $\hat{D}$.

The estimator $\hat{D}$ captures the difference between the true and reference residual distributions, which reflects the extent of model violations, making it instrumental in forming a model violations index (MVI). However, when more observations are used in regression, the value of $\hat{D}$ tends to increase logarithmically. This is because $D = \log(1 + D_{KL})$, and under the assumption of independence, D_{KL} is the sum of $D_{KL}^{(i)}$ across all observations. This does not mean that $\hat{D}$ becomes less reliable. In fact, larger samples often make model violations more visible in residual plots, unless strong overlapping masks the patterns.

However, to create a standardized and generalizable index, it is important to adjust for the effect of sample size. Therefore, the Model Violations Index (MVI) is proposed as

$$\text{MVI} = C + \hat{D} - \log(n), \tag{C1}$$

where C is a sufficiently large constant to ensure the result remains positive, and the $\log(n)$ term offset the increase in D with larger sample sizes.

Figure C1 displays the residual plots for fitted models exhibiting varying degrees of non-linearity and heteroskedasticity. Each residual plot's MVI is computed using Equation C1 with $C = 10$. When MVI > 8, the visual patterns are notably strong and easily discernible by humans. In the range $6 <$ MVI < 8, the visibility of the visual pattern diminishes as MVI decreases. Conversely, when MVI < 6, the visual pattern tends to become relatively faint and challenging to observe. Table C1 provides a summary of the MVI usage and it is applicable to other linear regression models.

Table C1. Degree of model violations or the strength of the visual signals according to the Model Violations Index (MVI). The constant C is set to be 10.

Degree of model violations	Range ($C = 10$)
Strong	$\text{MVI} > 8$
Moderate	$6 < \text{MVI} < 8$
Weak	$\text{MVI} < 6$

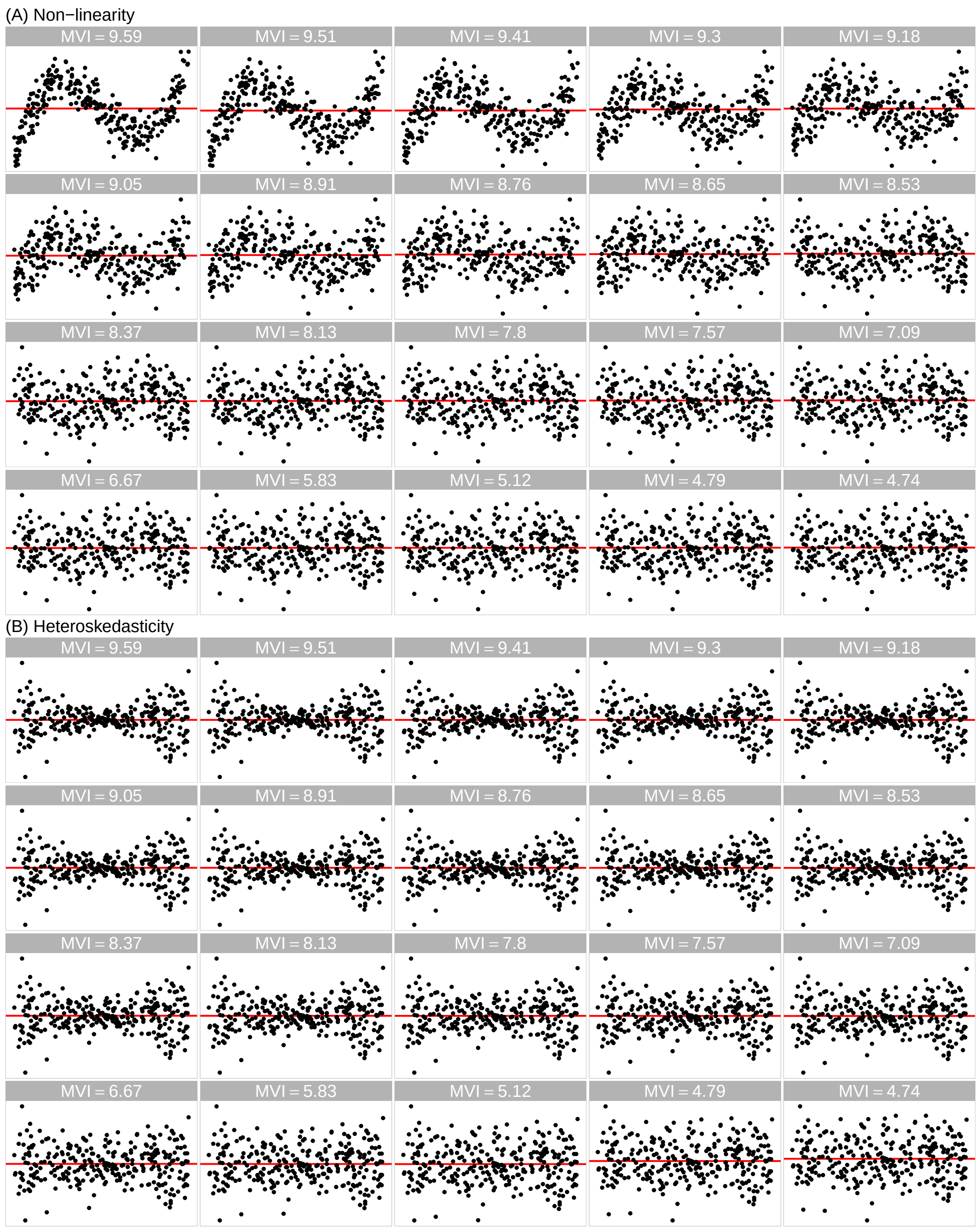

Figure C1. Residual plots generated from fitted models exhibiting varying degrees of (A) non-linearity and (B) heteroskedasticity violations. The model violations index (MVI) is displayed atop each residual plot. The non-linearity patterns are relatively strong for $MVI > 8$, and relatively weak for $MVI < 6$, while the heteroskedasticity patterns are relatively strong for $MVI > 8$, and relatively weak for $MVI < 6$.

References

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, et al. 2016. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." *arXiv preprint arXiv:1603.04467* .

Chollet, François, et al. 2015. "Keras." `https://keras.io`.

Chollet, Francois. 2021. *Deep learning with Python.* Simon and Schuster.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning.* MIT press.

Goscinski, Wojtek J, Paul McIntosh, Ulrich Felzmann, Anton Maksimenko, Christopher J Hall, Timur Gureyev, Darren Thompson, et al. 2014. "The multi-modal Australian ScienceS Imaging and Visualization Environment (MASSIVE) high performance computing infrastructure: applications in neuroscience and neuroinformatics research." *Frontiers in Neuroinformatics* 8: 30.

Hermite, M. 1864. *Sur un nouveau développement en série des fonctions.* Imprimerie de Gauthier-Villars.

Li, Weihao, Dianne Cook, Emi Tanaka, and Susan VanderPlas. 2024. "A plot is worth a thousand tests: Assessing residual diagnostics with the lineup protocol." *Journal of Computational and Graphical Statistics* 1–19.

Nair, Vinod, and Geoffrey E Hinton. 2010. "Rectified linear units improve restricted boltzmann machines." In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814.

O'Malley, Tom, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. 2019. "Keras Tuner." `https://github.com/keras-team/keras-tuner`.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. "Dropout: a simple way to prevent neural networks from overfitting." *The journal of machine learning research* 15 (1): 1929–1958.