Electronic Supplementary Information for: Determining charge transport regimes in organic molecular crystals: a machine learning framework

Tiago S. A. Cassiano[1], Marcelo L. Pereira Junior[2], Pedro H. de Oliveira Neto[1,3*] and Luiz A. Ribeiro Junior[1,4]

[1] *University of Brasília, Institute of Physics, 70.910-900, Brasília, Brazil*
[2] *University of Brasília, College of Technology, Department of Electrical Engineering, 70910-900, Brasília, Brazil.*
[3] *International Center of Physics, University of Brasília, 70919-970, Brazil.*
[4] *Computational Materials Laboratory, LCCMat, Institute of Physics, University of Brasília, 70910-900, Brasília, Brazil.*

*Corresponding Author: pedrohenrique@unb.br

Contents

1 Derivation of the Expression for the Current

The proof for Eq. 14 is equivalent to traditional tight-binding models. Here, we provide its derivation in the 1D case for a better readability of our methodology. The electronic eigenstate of the Hamiltonian, under a general tight-binding, is a linear combination of localized states, that is:

$$|\psi\rangle = \sum_n D_n(t)\,|n\rangle\,, \tag{S1}$$

where the components $D_n(t)$ change in time. The state evolves according to the time-dependent Schroedinger equation. In other words,

$$i\hbar\frac{\partial D_n}{\partial t} = \sum_{n'} H_{n',n}D_{n'}, \tag{S2}$$

in which $H_{n',n} = \langle n'|H|n\rangle$.

In a general tight-binding approach, only the elements $H_{n,n}$, $H_{n\mp1,n}$ are non-zero. Therefore, the expression can be rewritten as

$$i\hbar\frac{\partial D_n}{\partial t} = H_{n,n+1}D_{n+1} + H_{n,n-1}D_{n-1} + H_{n,n}D_n, \tag{S3}$$

with the corresponding complex conjugate

$$-i\hbar\frac{\partial D_n}{\partial t} = H^*_{n,n+1}D^*_{n+1} + H^*_{n,n-1}D^*_{n-1} + H_{n,n}D^*_n. \tag{S4}$$

We recall that the discrete continuity equation for the density at the n-th site, ρ_n, is

$$\frac{\partial \rho_n}{\partial t} = |e|D_n\frac{\partial D^*_n}{\partial t} + |e|D^*_n\frac{\partial D_n}{\partial t}. \tag{S5}$$

Substituting Eq. S3 and S4 into S6 returns

$$\frac{\partial \rho_n}{\partial t} = (-i|e|/\hbar)(H_{n,n+1}D_{n+1}D^*_n - H^*_{n+1,n}D^*_{n+1}D_n) - (-i|e|/\hbar)(H^*_{n-1,n}D_{n-1}D_n - H_{n,n-1}D_{n-1}D^*_n), \tag{S6}$$

where we recognize the charge density current flux

$$j_{n+1,n} = (-|e|/\hbar)(H_{n,n+1}D_{n+1}D^*_n - H^*_{n+1,n}D^*_{n+1}D_n), \tag{S7}$$

and influx

$$j_{n,n-1} = (-|e|/\hbar)(H^*_{n-1,n}D_{n-1}D_n - H_{n,n-1}D_{n-1}D^*_n). \tag{S8}$$

Note that $j_{n+1,n}$ is the sum of a complex number minus its conjugate. Therefore, we can further simplify $j_{n+1,n}$ into

$$j_{n+1,n} = -(2|e|/\hbar)\mathrm{Im}(H_{n,n+1}D_{n+1}D^*_n). \tag{S9}$$

2 Image pre-processing

An important step for a functional pipeline is the appropriate image pre-processing. Here, we standarized all heatmaps, initially compressing them to the size of 256x256 and applying a standard scaling filter. Moreover, their horizontal and vertical axes had a fixed scale. All images were generated using a gray-scale color palette to avoid noise during the training step.

3 Autoencoder Architecture

The encoder consists of four blocks of convolution layers, as shown in Figure S1. The first block has two convolution layers with 32 filters. Batch normalization is applied after each convolution[1]. At the end of the first block, a pooling filter is applied to compress the data. Then, a dropout layer is placed to avoid overfitting[1]. Blocks 2 and 3 follow a similar architecture using 64 and 128 filters. The last block is the bottleneck-dense layer, with 256 filters, renormalization, and dropout. The rectified linear unit (relu) function activated the neuron. The decoder structure is shown in Figure S1. It is worth mentioning that the number of filters in its last convolution layer is set based on the number of channels (which color ranges are allowed) in the original image (Nc). Moreover, this particular layer has the sigmoid as the activation function. Finally, we used the binary cross entropy as the training loss[1]. The autoencoder pipeline was implemented using the Python package Keras version 2.13.1.

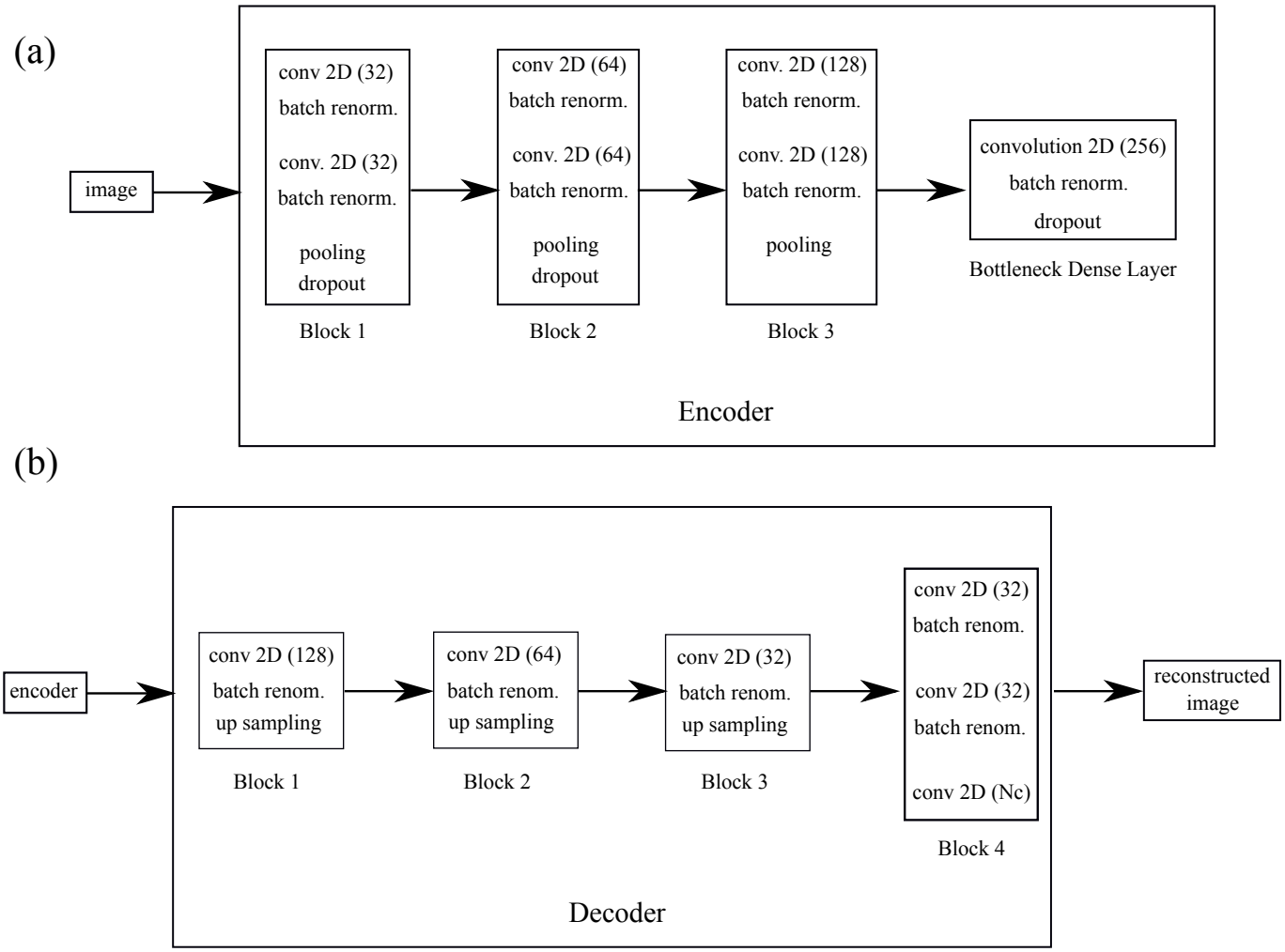

Figure S1: Encoder (a) and decoder (b) architectures used to process the charge density heatmap images.

We had used as optimizer algorithm the gradient descent Adam method and the binary cross entropy function as loss function. The learning rate and other parameters involving the optimizer were set using the Keras default values. Therefore, the learning rate was set to 0.001.

Moreover, due to the large size of the dataset, the images are loaded in the pipeline through batches. Here, we set the batch size equal to 40. Finally, the early stopping routine was active to avoid overfitting.

3.1 CNN validation

To ensure an appropriate image representation, the loss function progression was monitored, as shown in Figure S2 where the loss function is presented as a function of the number of epochs for the training and testing subsets. As can be seen, the two curves begin to converge after five epochs. Then, they both decay together until reaching a satisfactory value.

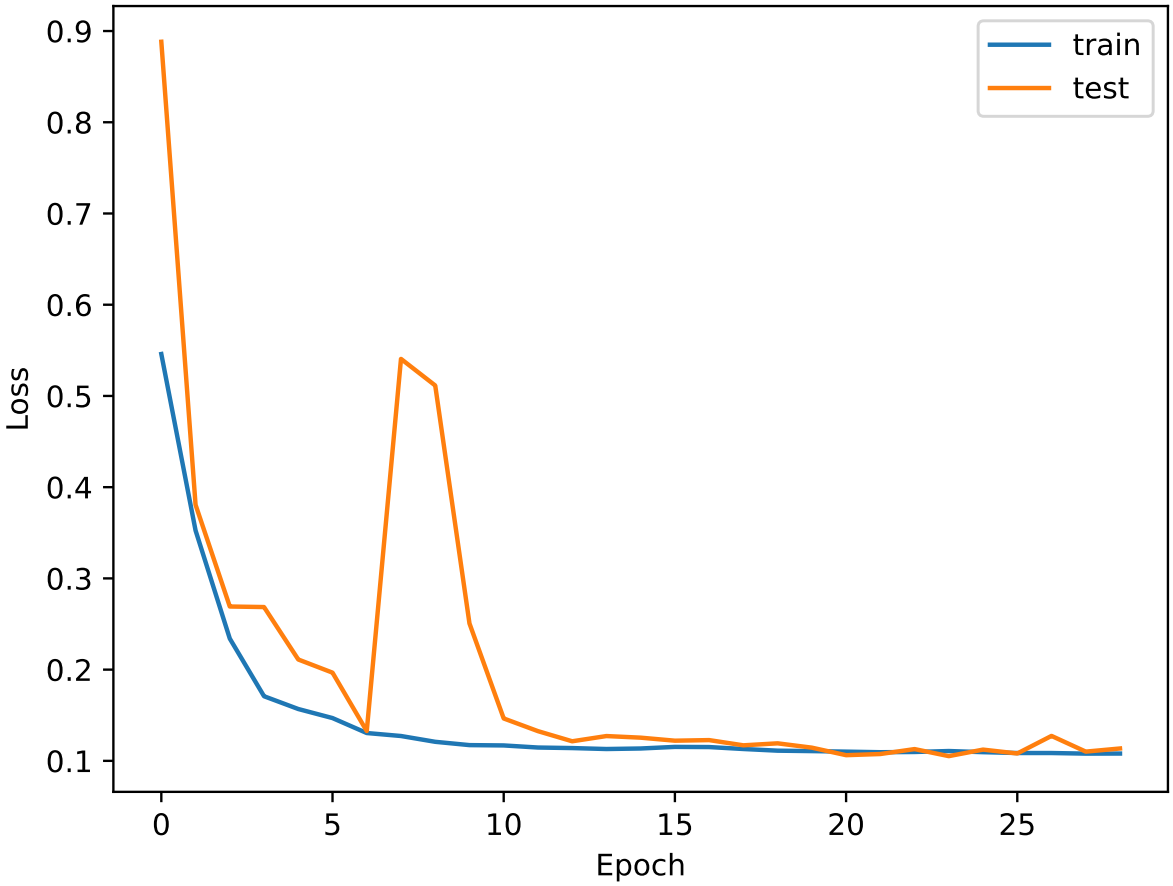

Figure S2: Loss function (binary cross entropy) as a function of the number of epochs for the train and test datasets.

To further ensure that the CNN pipeline preserves the images characteristics, we further compared the original and reconstructed images (after passing through the autoencoder workflow) of selected cases. These images are present in Figure S3.

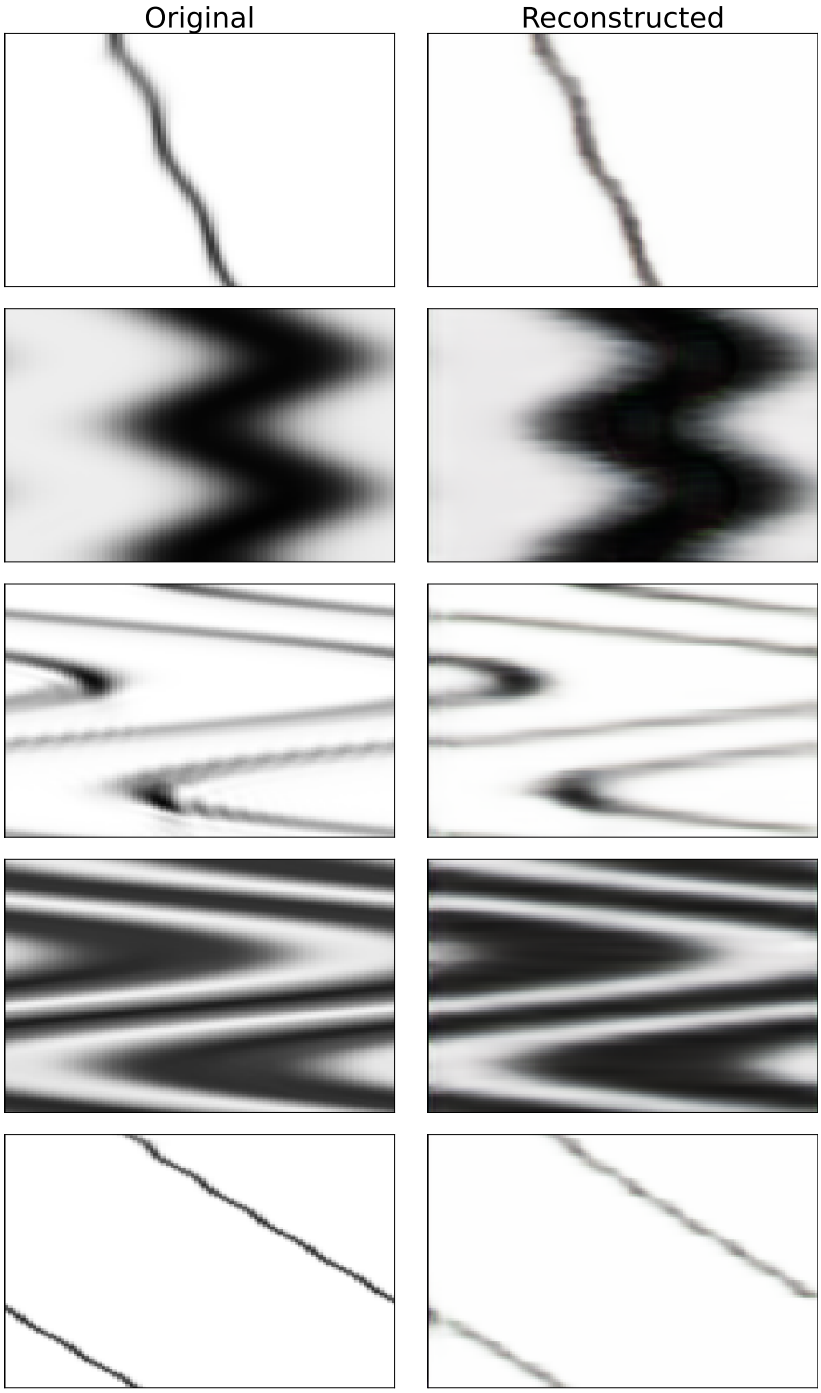

Figure S3: Selected examples where the original images, derived from the simulations, are compared with their reconstructed counterparts. The reconstructed images were obtained after applying the full autoenconder pipeline in the original image.

4 TSNE Parametrization

The representation of the dataset via the tsne method was made after a preliminary search for the appropriate parametrization tha best represents the clusterization. The variations from each parameters were: perplexity (5, 10, 40, 100, 500), number of iterations (300, 1500), learning rate (50, 200, 300), and early exaggeration (10, 100, 200). These limits were preselected after a qualitative search was made in which it was found

that, beyond these thresholds, the dataset representation showed not appropriate separation. Within this variation, we found that the following combination delivered the best separation is the one expressed in table S1.

Parameter	**Value**
Perplexity	10
Number of iterations	1500
Learning rate	200
Early exaggeration	100

Table S1: TSNE hyperparameters to generate Figure 3(a).

5 FFT mean frequency pairplot

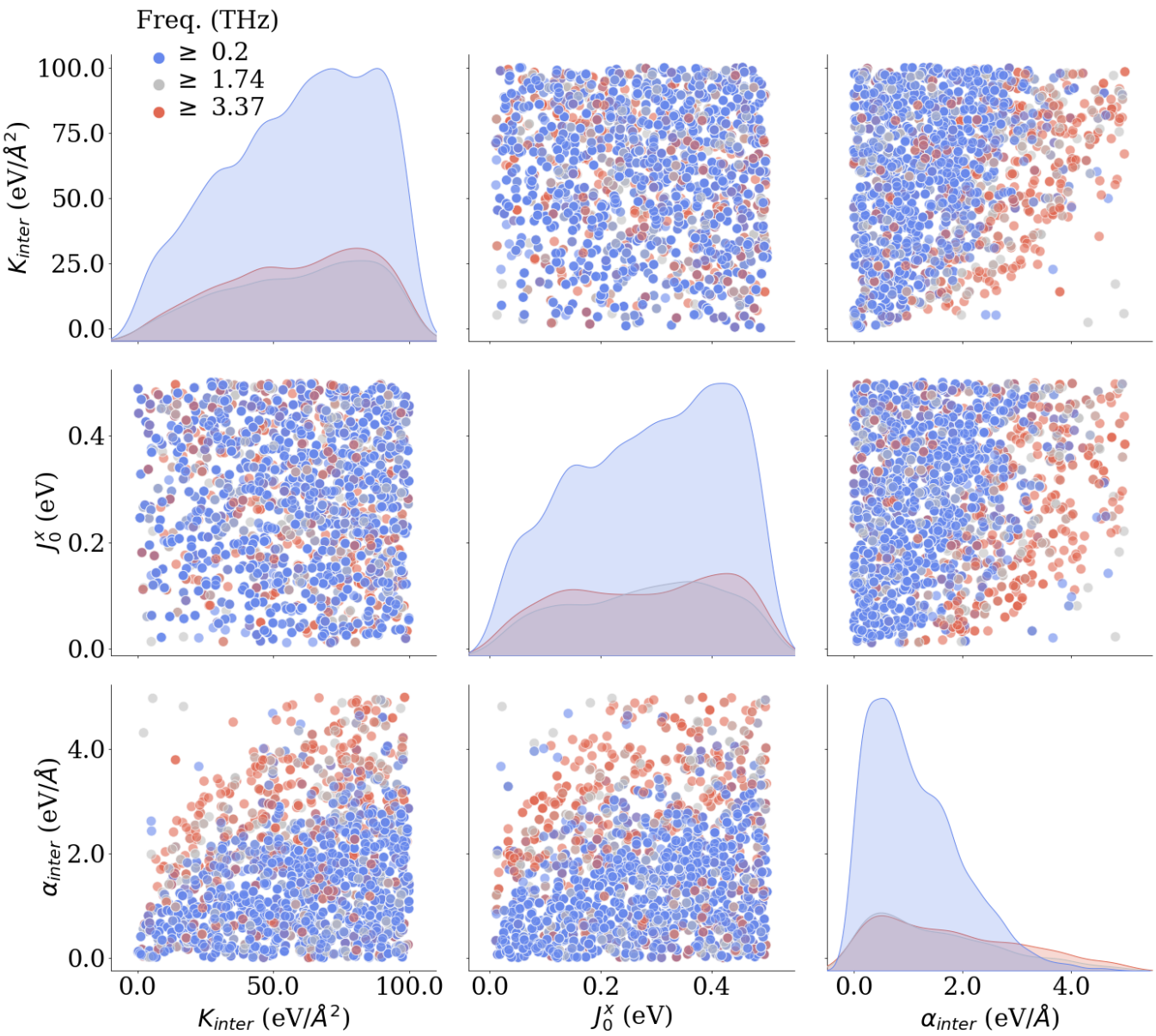

Figure S4: FFT mean frequency pairplot.

6 Δj_{max} pairplot

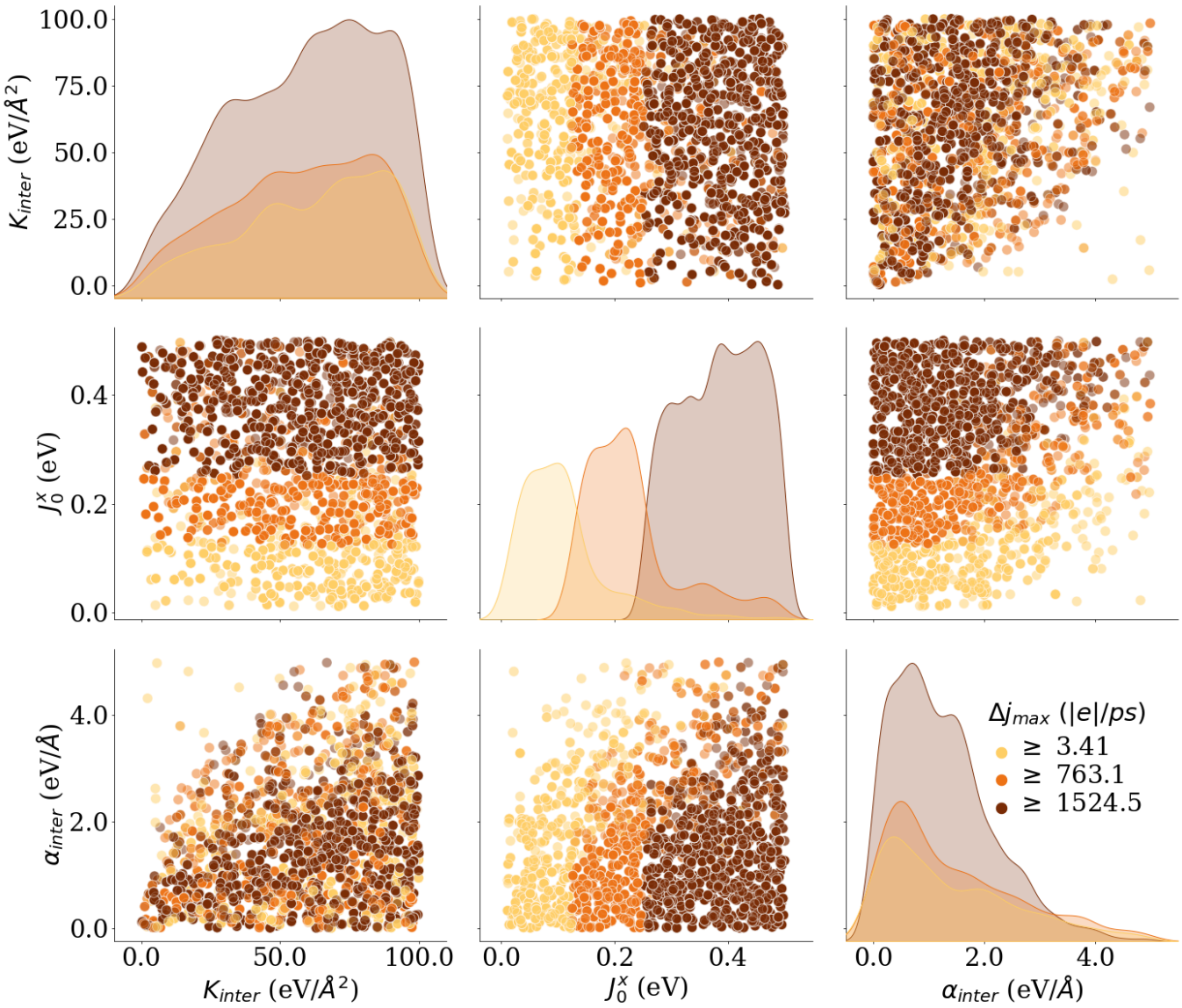

Figure S5: Charge current amplitude pairplot.

7 DC pairplot

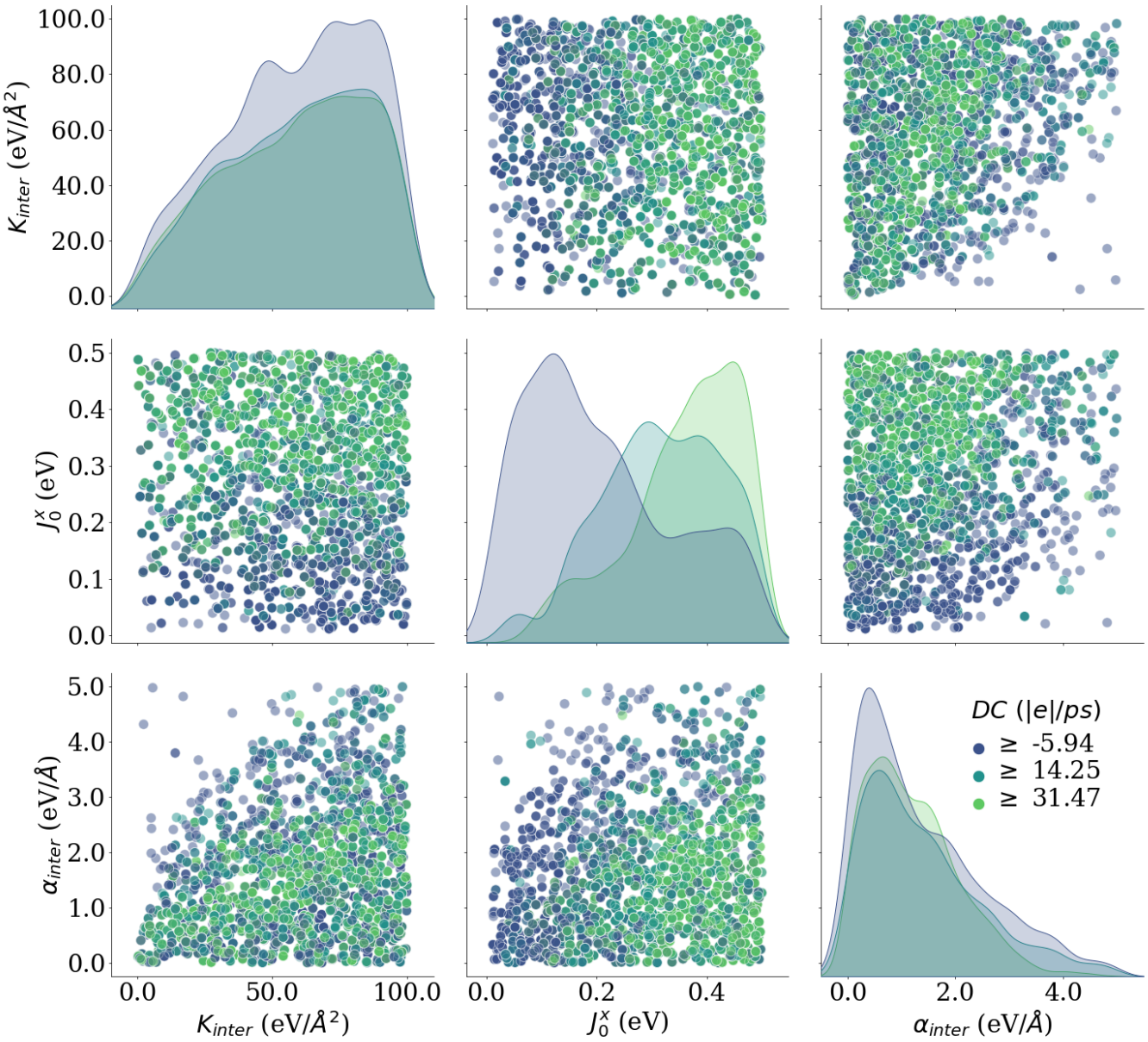

Figure S6: DC pairplot.

References

[1] François Chollet et al. Keras. `https://keras.io`, 2015.