

Projeto de Detecção de falha em Módulos Fotovoltaicos com PyTorch.

1ª parte do Projeto Final para a disciplina PPGEEC2318 - APRENDIZADO DE MÁQUINA.

Discentes: Guilherme Nascimento da Silva; Israel da Silva Félix de Lima.

Importação das bibliotecas

```
1 import kagglehub
2 from PIL import Image
3 import os
4 import cv2
5 import shutil
6 import pandas as pd
7 import numpy as np
8 import random
9 import matplotlib.pyplot as plt
10 import seaborn as sns
11 from sklearn.model_selection import train_test_split
12 from sklearn.preprocessing import LabelEncoder
13 from sklearn.preprocessing import StandardScaler
14 from sklearn.metrics import confusion_matrix
15 import matplotlib.image as mpimg
16
17 # Import PyTorch core and utilities for deep learning
18 import torch
19 import torch.optim as optim # Optimization algorithms
20 import torch.nn as nn # Neural network modules
21 import torch.nn.functional as F # Functional API for non-parametric operations
22
23 # Import PyTorch utilities for data loading and transformations
24 from torch.utils.data import DataLoader, Dataset, random_split, WeightedRandomSampler
25 from torchvision.transforms.v2 import Compose, ToImage, Normalize, ToPILImage, Resize, ToDtype
26
27 # Import dataset handling and learning rate schedulers
28 from torchvision.datasets import ImageFolder
29 from torch.optim.lr_scheduler import StepLR, ReduceLROnPlateau, MultiStepLR, CyclicLR, LambdaLR
```

Importação do Dataset

```
1 # Install dependencies as needed:
2 ! pip install kagglehub[pandas-datasets]
3 from kagglehub import KaggleDatasetAdapter
```

→ Mostrar saída oculta

```
1 # Download the dataset and get the path to the files
2 path = kagglehub.dataset_download("marcosgabriel/infrared-solar-modules")
3
4 # Construct the full path to the JSON file
5 file_path = os.path.join(path, "2020-02-14_InfraredSolarModules", "InfraredSolarModules", "module_metadata.json")
6
7 # Load the JSON file into a pandas DataFrame
8 df = pd.read_json(path_or_buf=file_path, orient='index').sort_index()
9
10 df = df.assign(image_filepath=df.image_filepath.apply(lambda x:os.path.join('/kaggle/input/infrared-solar-modules/2020-02-14_InfraredSolarModules', x)))
11
12 classification_type = 'anomaly_class';df
13
14
15 #print("First 5 records:", df.head())
```

```
image_filepath  anomaly_class
0  /kaggle/input/infrared-solar-modules/2020-02-1...  Offline-Module
1  /kaggle/input/infrared-solar-modules/2020-02-1...  Offline-Module
2  /kaggle/input/infrared-solar-modules/2020-02-1...  Offline-Module
3  /kaggle/input/infrared-solar-modules/2020-02-1...  Offline-Module
4  /kaggle/input/infrared-solar-modules/2020-02-1...  Offline-Module
...
19995 /kaggle/input/infrared-solar-modules/2020-02-1...  No-Anomaly
19996 /kaggle/input/infrared-solar-modules/2020-02-1...  No-Anomaly
19997 /kaggle/input/infrared-solar-modules/2020-02-1...  No-Anomaly
19998 /kaggle/input/infrared-solar-modules/2020-02-1...  No-Anomaly
19999 /kaggle/input/infrared-solar-modules/2020-02-1...  No-Anomaly
```

20000 rows × 2 columns

```
1 df.anomaly_class.value_counts()
```

anomaly_class	count
No-Anomaly	10000
Cell	1877
Vegetation	1639
Diode	1499
Cell-Multi	1288
Shadowing	1056
Cracking	940
Offline-Module	827
Hot-Spot	249
Hot-Spot-Multi	246
Soiling	204
Diode-Multi	175

dtype: int64

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 20000 entries, 0 to 19999
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   image_filepath  20000 non-null  object 
 1   anomaly_class   20000 non-null  object 
dtypes: object(2)
memory usage: 468.8+ KB
```

```
1 df.to_csv("Infrared_Solar_Modules_data.csv", index=False)
```

▼ Análise Exploratória de Dados (EDA)

Análise dados copiada de <https://www.kaggle.com/code/aliakbaryaghoubi/solar-modules-fault-detection-with-cnn-pytorch>

```
1 def plot(anomaly_type='all', plot_type='jet'):
2
3     if anomaly_type == 'all':
4         sample = df.groupby('anomaly_class').sample()
5     else:
6         sample = df[df['anomaly_class']==anomaly_type].sample(12)
7
8     plt.figure(figsize=(18,18))
9     for i, (image_path, class_name) in enumerate(sample.values, start=1):
```

```

10     image = cv2.imread(image_path, 0)
11
12     plt.subplot(2, 6, i)
13     plt.imshow(image, cmap=plot_type)
14     plt.subplots_adjust(top=0.52, hspace=0.2)
15     plt.title(class_name, fontsize=20)
16     plt.axis('off')
17     plt.show()

```

Architecture class

```

1 class Architecture(object):
2     def __init__(self, model, loss_fn, optimizer):
3         # Here we define the attributes of our class
4
5         # We start by storing the arguments as attributes
6         # to use them later
7         self.model = model
8         self.loss_fn = loss_fn
9         self.optimizer = optimizer
10        self.device = 'cuda' if torch.cuda.is_available() else 'cpu'
11        # Let's send the model to the specified device right away
12        self.model.to(self.device)
13
14        # These attributes are defined here, but since they are
15        # not informed at the moment of creation, we keep them None
16        self.train_loader = None
17        self.val_loader = None
18
19        # These attributes are going to be computed internally
20        self.losses = []
21        self.val_losses = []
22        self.total_epochs = 0
23
24        # Creates the train_step function for our model,
25        # loss function and optimizer
26        # Note: there are NO ARGUMENTS there! It makes use of the class
27        # attributes directly
28        self.train_step_fn = self._make_train_step_fn()
29        # Creates the val_step function for our model and loss
30        self.val_step_fn = self._make_val_step_fn()
31
32        # for hook purposes
33        self.handles = {}
34        self.visualization = {}
35
36    def to(self, device):
37        # This method allows the user to specify a different device
38        # It sets the corresponding attribute (to be used later in
39        # the mini-batches) and sends the model to the device
40        try:
41            self.device = device
42            self.model.to(self.device)
43        except RuntimeError:
44            self.device = 'cuda' if torch.cuda.is_available() else 'cpu'
45            print(f"Couldn't send it to {device}, sending it to {self.device} instead.")
46            self.model.to(self.device)
47
48    def set_loaders(self, train_loader, val_loader=None):
49        # This method allows the user to define which train_loader (and val_loader, optionally) to use
50        # Both loaders are then assigned to attributes of the class
51        # So they can be referred to later
52        self.train_loader = train_loader
53        self.val_loader = val_loader
54
55    def _make_train_step_fn(self):
56        # This method does not need ARGUMENTS... it can refer to
57        # the attributes: self.model, self.loss_fn and self.optimizer
58
59        # Builds function that performs a step in the train loop
60        def perform_train_step_fn(x, y):
61            # Sets model to TRAIN mode
62            self.model.train()
63
64            # Step 1 - Computes our model's predicted output - forward pass
65            yhat = self.model(x)
66            # Step 2 - Computes the loss
67            loss = self.loss_fn(yhat, y)
68            # Step 3 - Computes gradients for both "a" and "b" parameters

```

```

69         loss.backward()
70         # Step 4 - Updates parameters using gradients and the learning rate
71         self.optimizer.step()
72         self.optimizer.zero_grad()
73
74     # Returns the loss
75     return loss.item()
76
77     # Returns the function that will be called inside the train loop
78     return perform_train_step_fn
79
80 def _make_val_step_fn(self):
81     # Builds function that performs a step in the validation loop
82     def perform_val_step_fn(x, y):
83         # Sets model to EVAL mode
84         self.model.eval()
85
86         # Step 1 - Computes our model's predicted output - forward pass
87         yhat = self.model(x)
88         # Step 2 - Computes the loss
89         loss = self.loss_fn(yhat, y)
90         # There is no need to compute Steps 3 and 4, since we don't update parameters during evaluation
91         return loss.item()
92
93     return perform_val_step_fn
94
95 def _mini_batch(self, validation=False):
96     # The mini-batch can be used with both loaders
97     # The argument `validation` defines which loader and
98     # corresponding step function is going to be used
99     if validation:
100         data_loader = self.val_loader
101         step_fn = self.val_step_fn
102     else:
103         data_loader = self.train_loader
104         step_fn = self.train_step_fn
105
106     if data_loader is None:
107         return None
108
109     # Once the data loader and step function, this is the same
110     # mini-batch loop we had before
111     mini_batch_losses = []
112     for x_batch, y_batch in data_loader:
113         x_batch = x_batch.to(self.device)
114         y_batch = y_batch.to(self.device)
115
116         mini_batch_loss = step_fn(x_batch, y_batch)
117         mini_batch_losses.append(mini_batch_loss)
118
119     loss = np.mean(mini_batch_losses)
120     return loss
121
122     # this function was updated in this class
123 def set_seed(self, seed=42):
124     torch.backends.cudnn.deterministic = True
125     torch.backends.cudnn.benchmark = False
126     torch.manual_seed(seed)
127     np.random.seed(seed)
128     random.seed(seed)
129     try:
130         self.train_loader.sampler.generator.manual_seed(seed)
131     except AttributeError:
132         pass
133
134 def train(self, n_epochs, seed=42):
135     # To ensure reproducibility of the training process
136     self.set_seed(seed)
137
138     for epoch in range(n_epochs):
139         # Keeps track of the numbers of epochs
140         # by updating the corresponding attribute
141         self.total_epochs += 1
142
143         # inner loop
144         # Performs training using mini-batches
145         loss = self._mini_batch(validation=False)
146         self.losses.append(loss)
147
148         # VALIDATION
149         # no gradients in validation!
150         with torch.no_grad():

```

```

151         # Performs evaluation using mini-batches
152         val_loss = self._mini_batch(validation=True)
153         self.val_losses.append(val_loss)
154
155     def save_checkpoint(self, filename):
156         # Builds dictionary with all elements for resuming training
157         checkpoint = {'epoch': self.total_epochs,
158                       'model_state_dict': self.model.state_dict(),
159                       'optimizer_state_dict': self.optimizer.state_dict(),
160                       'loss': self.losses,
161                       'val_loss': self.val_losses}
162
163         torch.save(checkpoint, filename)
164
165     def load_checkpoint(self, filename):
166         # Loads dictionary
167         checkpoint = torch.load(filename)
168
169         # Restore state for model and optimizer
170         self.model.load_state_dict(checkpoint['model_state_dict'])
171         self.optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
172
173         self.total_epochs = checkpoint['epoch']
174         self.losses = checkpoint['loss']
175         self.val_losses = checkpoint['val_loss']
176
177         self.model.train() # always use TRAIN for resuming training
178
179     def predict(self, x):
180         # Set is to evaluation mode for predictions
181         self.model.eval()
182         # Takes anumpy input and make it a float tensor
183         x_tensor = torch.as_tensor(x).float()
184         # Send input to device and uses model for prediction
185         y_hat_tensor = self.model(x_tensor.to(self.device))
186         # Set it back to train mode
187         self.model.train()
188         # Detaches it, brings it to CPU and back to Numpy
189         return y_hat_tensor.detach().cpu().numpy()
190
191     def count_parameters(self):
192         return sum(p.numel() for p in self.model.parameters() if p.requires_grad)
193
194     def plot_losses(self):
195         fig = plt.figure(figsize=(10, 4))
196         plt.plot(self.losses, label='Training Loss', c='b')
197         plt.plot(self.val_losses, label='Validation Loss', c='r')
198         plt.yscale('log')
199         plt.xlabel('Epochs')
200         plt.ylabel('Loss')
201         plt.legend()
202         plt.tight_layout()
203         return fig
204
205     @staticmethod
206     def _visualize_tensors(axes, x, y=None, yhat=None, layer_name='', title=None):
207         # The number of images is the number of subplots in a row
208         n_images = len(axes)
209         # Gets max and min values for scaling the grayscale
210         minv, maxv = np.min(x[:n_images]), np.max(x[:n_images])
211         # For each image
212         for j, image in enumerate(x[:n_images]):
213             ax = axes[j]
214             # Sets title, labels, and removes ticks
215             if title is not None:
216                 ax.set_title(f'{title} #{j}', fontsize=12)
217                 shp = np.atleast_2d(image).shape
218                 ax.set_ylabel(
219                     f'{layer_name}\n{shp[0]}x{shp[1]}',
220                     rotation=0, labelpad=40
221                 )
222                 xlabel1 = '' if y is None else f'\nLabel: {y[j]}'
223                 xlabel2 = '' if yhat is None else f'\nPredicted: {yhat[j]}'
224                 xlabel = f'{xlabel1}{xlabel2}'
225                 if len(xlabel):
226                     ax.set_xlabel(xlabel, fontsize=12)
227                     ax.set_xticks([])
228                     ax.set_yticks([])
229
230                 # Plots weight as an image
231                 ax.imshow(
232                     np.atleast_2d(image.squeeze()),
```

```

233         cmap='gray',
234         vmin=minv,
235         vmax=maxv
236     )
237     return
238
239 def visualize_filters(self, layer_name, **kwargs):
240     try:
241         # Gets the layer object from the model
242         layer = self.model
243         for name in layer_name.split('.'):
244             layer = getattr(layer, name)
245         # We are only looking at filters for 2D convolutions
246         if isinstance(layer, nn.Conv2d):
247             # Takes the weight information
248             weights = layer.weight.data.cpu().numpy()
249             # weights -> (channels_out (filter), channels_in, H, W)
250             n_filters, n_channels, _, _ = weights.shape
251
252             # Builds a figure
253             size = (2 * n_channels + 2, 2 * n_filters)
254             fig, axes = plt.subplots(n_filters, n_channels,
255                                     figsize=size)
256             axes = np.atleast_2d(axes)
257             axes = axes.reshape(n_filters, n_channels)
258             # For each channel_out (filter)
259             for i in range(n_filters):
260                 Architecture._visualize_tensors(
261                     axes[i, :],
262                     weights[i],
263                     layer_name=f'Filter #{i}',
264                     title='Channel'
265                 )
266
267             for ax in axes.flat:
268                 ax.label_outer()
269
270             fig.tight_layout()
271             return fig
272     except AttributeError:
273         return
274
275 def attach_hooks(self, layers_to_hook, hook_fn=None):
276     # Clear any previous values
277     self.visualization = {}
278     # Creates the dictionary to map layer objects to their names
279     modules = list(self.model.named_modules())
280     layer_names = {layer: name for name, layer in modules[1:]}
281
282     if hook_fn is None:
283         # Hook function to be attached to the forward pass
284         def hook_fn(layer, inputs, outputs):
285             # Gets the layer name
286             name = layer_names[layer]
287             # Detaches outputs
288             values = outputs.detach().cpu().numpy()
289             # Since the hook function may be called multiple times
290             # for example, if we make predictions for multiple mini-batches
291             # it concatenates the results
292             if self.visualization[name] is None:
293                 self.visualization[name] = values
294             else:
295                 self.visualization[name] = np.concatenate([self.visualization[name], values])
296
297         for name, layer in modules:
298             # If the layer is in our list
299             if name in layers_to_hook:
300                 # Initializes the corresponding key in the dictionary
301                 self.visualization[name] = None
302                 # Register the forward hook and keep the handle in another dict
303                 self.handles[name] = layer.register_forward_hook(hook_fn)
304
305     def remove_hooks(self):
306         # Loops through all hooks and removes them
307         for handle in self.handles.values():
308             handle.remove()
309         # Clear the dict, as all hooks have been removed
310         self.handles = {}
311
312     def visualize_outputs(self, layers, n_images=10, y=None, yhat=None):
313         layers = filter(lambda l: l in self.visualization.keys(), layers)
314         layers = list(layers)

```

```

315     shapes = [self.visualization[layer].shape for layer in layers]
316     n_rows = [shape[1] if len(shape) == 4 else 1
317               for shape in shapes]
318     total_rows = np.sum(n_rows)
319
320     fig, axes = plt.subplots(total_rows, n_images,
321                             figsize=(1.5*n_images, 1.5*total_rows))
322     axes = np.atleast_2d(axes).reshape(total_rows, n_images)
323
324     # Loops through the layers, one layer per row of subplots
325     row = 0
326     for i, layer in enumerate(layers):
327         start_row = row
328         # Takes the produced feature maps for that layer
329         output = self.visualization[layer]
330
331         is_vector = len(output.shape) == 2
332
333         for j in range(n_rows[i]):
334             Architecture._visualize_tensors(
335                 axes[row, :],
336                 output if is_vector else output[:, j].squeeze(),
337                 y,
338                 yhat,
339                 layer_name=layer_name[i] \
340                   if is_vector \
341                   else f'{layer_name[i]}\nfil#{row-start_row}',
342                 title='Image' if (row == 0) else None
343             )
344             row += 1
345
346     for ax in axes.flat:
347         ax.label_outer()
348
349     plt.tight_layout()
350     return fig
351
352 def correct(self, x, y, threshold=.5):
353     self.model.eval()
354     yhat = self.model(x.to(self.device))
355     y = y.to(self.device)
356     self.model.train()
357
358     # We get the size of the batch and the number of classes
359     # (only 1, if it is binary)
360     n_samples, n_dims = yhat.shape
361     if n_dims > 1:
362         # In a multiclass classification, the biggest logit
363         # always wins, so we don't bother getting probabilities
364
365         # This is PyTorch's version of argmax,
366         # but it returns a tuple: (max value, index of max value)
367         _, predicted = torch.max(yhat, 1)
368     else:
369         n_dims += 1
370         # In binary classification, we NEED to check if the
371         # last layer is a sigmoid (and then it produces probs)
372         if isinstance(self.model, nn.Sequential) and \
373             isinstance(self.model[-1], nn.Sigmoid):
374             predicted = (yhat > threshold).long()
375             # or something else (logits), which we need to convert
376             # using a sigmoid
377         else:
378             predicted = (F.sigmoid(yhat) > threshold).long()
379
380     # How many samples got classified correctly for each class
381     result = []
382     for c in range(n_dims):
383         n_class = (y == c).sum().item()
384         n_correct = (predicted[y == c] == c).sum().item()
385         result.append((n_correct, n_class))
386     return torch.tensor(result)
387
388
389     @staticmethod
390     def loader_apply(loader, func, reduce='sum'):
391         results = [func(x, y) for i, (x, y) in enumerate(loader)]
392         results = torch.stack(results, axis=0)
393
394         if reduce == 'sum':
395             results = results.sum(axis=0)
396         elif reduce == 'mean':

```

```

397         results = results.float().mean(axis=0)
398
399     return results
400
401 @staticmethod
402 def statistics_per_channel(images, labels):
403     # NCHW
404     n_samples, n_channels, n_height, n_weight = images.size()
405     # Flatten HW into a single dimension
406     flatten_per_channel = images.reshape(n_samples, n_channels, -1)
407
408     # Computes statistics of each image per channel
409     # Average pixel value per channel
410     # (n_samples, n_channels)
411     means = flatten_per_channel.mean(axis=2)
412     # Standard deviation of pixel values per channel
413     # (n_samples, n_channels)
414     stds = flatten_per_channel.std(axis=2)
415
416     # Adds up statistics of all images in a mini-batch
417     # (1, n_channels)
418     sum_means = means.sum(axis=0)
419     sum_stds = stds.sum(axis=0)
420     # Makes a tensor of shape (1, n_channels)
421     # with the number of samples in the mini-batch
422     n_samples = torch.tensor([n_samples]*n_channels).float()
423
424     # Stack the three tensors on top of one another
425     # (3, n_channels)
426     return torch.stack([n_samples, sum_means, sum_stds], axis=0)
427
428 @staticmethod
429 def make_normalizer(loader):
430     total_samples, total_means, total_stds = Architecture.loader_apply(loader, Architecture.statistics_per_channel)
431     norm_mean = total_means / total_samples
432     norm_std = total_stds / total_samples
433     return Normalize(mean=norm_mean, std=norm_std)
434
435 def lr_range_test(self, data_loader, end_lr, num_iter=100, step_mode='exp', alpha=0.05, ax=None):
436     # Since the test updates both model and optimizer we need to store
437     # their initial states to restore them in the end
438     previous_states = {'model': deepcopy(self.model.state_dict()),
439                         'optimizer': deepcopy(self.optimizer.state_dict())}
440     # Retrieves the learning rate set in the optimizer
441     start_lr = self.optimizer.state_dict()['param_groups'][0]['lr']
442
443     # Builds a custom function and corresponding scheduler
444     lr_fn = make_lr_fn(start_lr, end_lr, num_iter)
445     scheduler = LambdaLR(self.optimizer, lr_lambda=lr_fn)
446
447     # Variables for tracking results and iterations
448     tracking = {'loss': [], 'lr': []}
449     iteration = 0
450
451     # If there are more iterations than mini-batches in the data loader,
452     # it will have to loop over it more than once
453     while (iteration < num_iter):
454         # That's the typical mini-batch inner loop
455         for x_batch, y_batch in data_loader:
456             x_batch = x_batch.to(self.device)
457             y_batch = y_batch.to(self.device)
458             # Step 1
459             yhat = self.model(x_batch)
460             # Step 2
461             loss = self.loss_fn(yhat, y_batch)
462             # Step 3
463             loss.backward()
464
465             # Here we keep track of the losses (smoothed)
466             # and the learning rates
467             tracking['lr'].append(scheduler.get_last_lr()[0])
468             if iteration == 0:
469                 tracking['loss'].append(loss.item())
470             else:
471                 prev_loss = tracking['loss'][-1]
472                 smoothed_loss = alpha * loss.item() + (1-alpha) * prev_loss
473                 tracking['loss'].append(smoothed_loss)
474
475             iteration += 1
476             # Number of iterations reached
477             if iteration == num_iter:
478                 break

```

```

479
480         # Step 4
481         self.optimizer.step()
482         scheduler.step()
483         self.optimizer.zero_grad()
484
485     # Restores the original states
486     self.optimizer.load_state_dict(previous_states['optimizer'])
487     self.model.load_state_dict(previous_states['model'])
488
489     if ax is None:
490         fig, ax = plt.subplots(1, 1, figsize=(6, 4))
491     else:
492         fig = ax.get_figure()
493     ax.plot(tracking['lr'], tracking['loss'])
494     if step_mode == 'exp':
495         ax.set_xscale('log')
496     ax.set_xlabel('Learning Rate')
497     ax.set_ylabel('Loss')
498     fig.tight_layout()
499     return tracking, fig
500
501 def set_optimizer(self, optimizer):
502     self.optimizer = optimizer

```

▼ Ajuste dos Dados Para Teste e Treinamento

▼ Alocação das as imagens em pastas

Distribuição dos dados de treino e de teste

```

1 df_train, df_test = train_test_split(df, test_size=0.15, random_state=42)
2
3 print(f"Tamanho do dataset de treino: {len(df_train)}")
4 print(f"Tamanho do dataset de teste: {len(df_test)}")

```

→ Tamanho do dataset de treino: 17000
Tamanho do dataset de teste: 3000

```
1 df_test.anomaly_class.value_counts()
```

→

anomaly_class	count
No-Anomaly	1488
Cell	297
Vegetation	243
Cell-Multi	208
Diode	206
Shadowing	171
Cracking	144
Offline-Module	113
Hot-Spot-Multi	39
Hot-Spot	38
Soiling	34
Diode-Multi	19

```

anomaly_class
No-Anomaly    1488
Cell          297
Vegetation    243
Cell-Multi    208
Diode         206
Shadowing     171
Cracking      144
Offline-Module 113
Hot-Spot-Multi 39
Hot-Spot       38
Soiling        34
Diode-Multi    19

dtype: int64

```

```
1 df_test.head()
```

	image_filepath	anomaly_class
10650	/kaggle/input/infrared-solar-modules/2020-02-1...	No-Anomaly
2041	/kaggle/input/infrared-solar-modules/2020-02-1...	Diode
8668	/kaggle/input/infrared-solar-modules/2020-02-1...	Vegetation
1114	/kaggle/input/infrared-solar-modules/2020-02-1...	Diode
13902	/kaggle/input/infrared-solar-modules/2020-02-1...	No-Anomaly

Função de limpeza de dataset

```

1 def clear_dir(dir_path):
2 # Check if the directory exists
3     if os.path.exists(dir_path):
4         # Iterate over all items in the directory
5         for item in os.listdir(dir_path):
6             item_path = os.path.join(dir_path, item)
7             # If the item is a file, remove it
8             if os.path.isfile(item_path):
9                 os.remove(item_path)
10            # If the item is a directory, remove it and all its contents
11            elif os.path.isdir(item_path):
12                shutil.rmtree(item_path)
13    else:
14        print(f"Directory '{dir_path}' not found.")

```

Criação dos diretórios de teste e treinamento

Diretório de teste

```

1 output_test = "dataset_test"
2 # Check if the directory exists
3 if os.path.exists(output_test):
4     # If it exists, clear it
5     clear_dir(output_test)
6     # Then remove the directory itself
7     os.rmdir(output_test) # Use rmdir to remove the now empty directory
8 # Create the directory (it will be created if it didn't exist or was just removed)
9 os.makedirs(output_test)
10
11
12 # Iterate over the rows of the DataFrame and save each image
13 for index_test, row_test in df_test.iterrows():
14     image_path_test = row_test['image_filepath']
15     anomaly_class_test = row_test['anomaly_class']
16
17     # Check if the file exists before attempting to read
18     if not os.path.exists(image_path_test):
19         print(f"Warning: File not found at {image_path_test}. Skipping.")
20         continue
21
22     # Read the image
23     image_test = cv2.imread(image_path_test)
24
25     # Check if the image was loaded successfully
26     if image_test is None:
27         print(f"Warning: Could not read image from {image_path_test}. Skipping.")
28         continue # Skip to the next image if reading failed
29
30     # Create a subdirectory for each anomaly class
31     class_test = os.path.join(output_test, anomaly_class_test.replace(' ', '_'))
32     os.makedirs(class_test, exist_ok=True)
33
34     # Define the output filename
35     # Use the index as a unique identifier
36     output_filename_test = f"{index_test}.jpg"
37     output_filepath_test = os.path.join(class_test, output_filename_test)
38
39     # Save the image
40     cv2.imwrite(output_filepath_test, image_test)
41
42 print(f"Saved all images to the '{output_test}' directory.")

```

Saved all images to the 'dataset_test' directory.

Diretório de treinamento

```

1 output_train = "dataset_train"
2 # Check if the directory exists
3 if os.path.exists(output_train):
4     # If it exists, clear it
5     clear_dir(output_train)
6     # Then remove the directory itself
7     os.rmdir(output_train) # Use rmdir to remove the now empty directory
8 # Create the directory (it will be created if it didn't exist or was just removed)
9 os.makedirs(output_train, exist_ok=True)
10
11 for index_train, row_train in df_train.iterrows():
12     image_path_train = row_train['image_filepath']
13     anomaly_class_train = row_train['anomaly_class']
14
15     # Check if the file exists before attempting to read
16     if not os.path.exists(image_path_train):
17         print(f"Warning: File not found at {image_path_train}. Skipping.")
18         continue
19
20     # Read the image
21     image_train = cv2.imread(image_path_train)
22
23     # Check if the image was loaded successfully
24     if image_train is None:
25         print(f"Warning: Could not read image from {image_path_train}. Skipping.")
26         continue # Skip to the next image if reading failed
27
28
29     # Create a subdirectory for each anomaly class
30     class_train = os.path.join(output_train, anomaly_class_train.replace(' ', '_'))
31     os.makedirs(class_train, exist_ok=True)
32
33     # Define the output filename
34     # Use the index as a unique identifier
35     output_filename_train = f"{index_train}.jpg"
36     output_filepath_train = os.path.join(class_train, output_filename_train)
37
38     # Save the image
39     cv2.imwrite(output_filepath_train, image_train)
40
41 print(f"Saved all images to the '{output_train}' directory.")

```

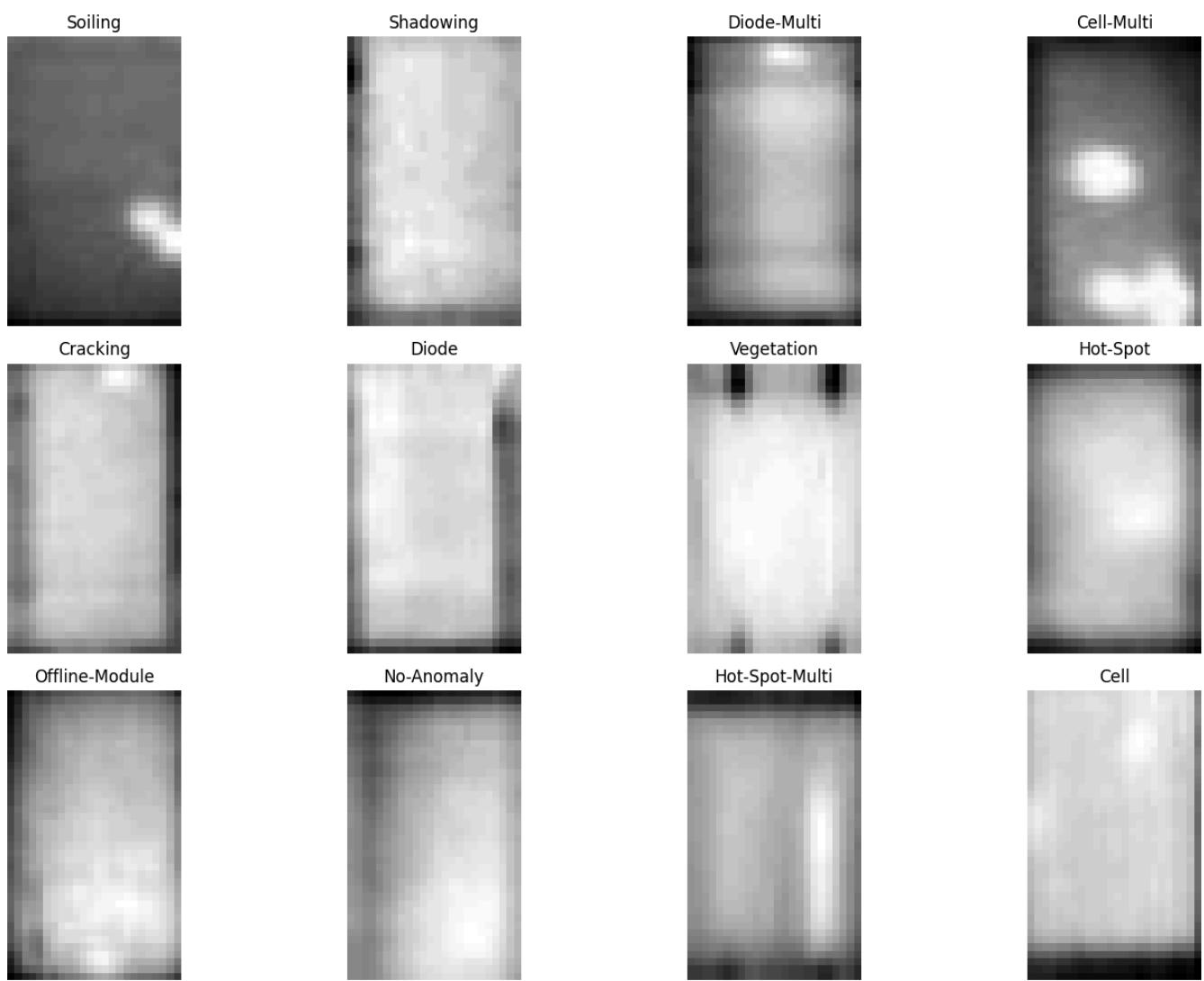
Saved all images to the 'dataset_train' directory.

Plotagem exemplo dos dados

```

1 # List the subdirectories (anomaly classes) in the test directory
2 test_subdirectories = [os.path.join("dataset_train", d) for d in os.listdir("dataset_train") if os.path.isdir(os.path.join("dataset_
3
4 plt.figure(figsize=(15, 10))
5 for i, subdir in enumerate(test_subdirectories):
6     # Get a list of all image files in the subdirectory
7     image_files = [f for f in os.listdir(subdir) if os.path.isfile(os.path.join(subdir, f))]
8
9     # Check if there are any images in the subdirectory
10    if image_files:
11        # Choose the first image file
12        image_path = os.path.join(subdir, image_files[0])
13
14        # Read the image
15        image = cv2.imread(image_path, 0)
16
17        # Get the anomaly class name from the directory name
18        anomaly_class = os.path.basename(subdir).replace('_', ' ')
19
20        plt.subplot(3, 4, i + 1) # Adjust subplot grid size as needed
21        plt.imshow(image, cmap='gray')
22        plt.title(anomaly_class, fontsize=12)
23        plt.axis('off')
24
25 plt.tight_layout()
26 plt.show()

```



▼ Preparação dos dados

▼ ImageFolder

```

1 # Compose a sequence of preprocessing transforms
2 # 1) Resize images to 46x46 pixels
3 # 2) Ensure output is a PIL/torchvision Image (dropping any alpha channel)
4 # 3) Convert pixel values to float32 and scale from [0-255] to [0.0-1.0]
5 temp_transform = Compose([
6     Resize((46,46)),           # Resize each image to 46x46
7     ToImage(),                # Convert tensor back to PIL Image (enforces RGB)
8     ToDtype(torch.float32, scale=True) # Cast to float32 and normalize pixel range
9 ])
10
11 # Create an ImageFolder dataset from the 'rps' directory
12 # Images are grouped by subfolder name as class labels, and each image is transformed
13 temp_dataset = ImageFolder(
14     root='dataset_train',
15     transform=temp_transform      # Apply the preprocessing pipeline to every image
16 )

```

```

1 # Get total number of samples in the dataset
2 dataset_size = len(temp_dataset)
3 print(f"Dataset size: {dataset_size} images")
4
5 # Get number of classes
6 num_classes = len(temp_dataset.classes)
7 print(f"Number of classes: {num_classes}")

→ Dataset size: 17000 images
Number of classes: 12

```

▼ Standardization

```

1 temp_loader = DataLoader(temp_dataset, batch_size=16)

1 # Each column represents a channel
2 # first row is the number of data points
3 # second row is the the sum of mean values
4 # third row is the sum of standard deviations
5 first_images, first_labels = next(iter(temp_loader))
6 Architecture.statistics_per_channel(first_images, first_labels)

→ tensor([[16.0000, 16.0000, 16.0000],
           [10.0997, 10.0997, 10.0997],
           [ 1.1843,  1.1843,  1.1843]])

1 # We can leverage the loader_apply() method to get the sums for the whole dataset:
2 results = Architecture.loader_apply(temp_loader, Architecture.statistics_per_channel)
3 results

→ tensor([[17000.0000, 17000.0000, 17000.0000],
           [10518.5723, 10518.5723, 10518.5723],
           [ 1247.9636, 1247.9636, 1247.9636]])

1 # we can compute the average mean value and the average standard deviation, per channel.
2 # Better yet, let's make it a method that takes a data loader and
3 # returns an instance of the Normalize() transform
4 normalizer = Architecture.make_normalizer(temp_loader)
5 normalizer

→ Normalize(mean=[tensor(0.6187), tensor(0.6187), tensor(0.6187)], std=[tensor(0.0734), tensor(0.0734), tensor(0.0734)],
           inplace=False)

```

▼ Aplicação das métricas de normalização nos datasets

```

1 # Define a pipeline of image transformations:
2 # 1) Redefine cada imagem para 46x46 pixels
3 # 2) Ensure the output is a PIL/torchvision image (dropping any alpha channel)
4 # 3) Cast pixels to float32 and scale from [0-255] to [0.0-1.0]
5 # 4) Apply the user-defined normalization (e.g., mean/std normalization)
6 composer = Compose([
7     Resize((46,46)),                      # Resize to 46x46
8     ToImage(),                            # Convert to PIL Image in RGB
9     ToDtype(torch.float32, scale=True),    # Cast to float32 and normalize to [0,1]
10    normalizer                           # Apply custom normalization transform
11 ])
12
13 # Instantiate training and validation datasets from folders:
14 # - 'dataset_train' contains subfolders per class for training
15 # - 'dataset_test' likewise for validation
16 train_data = ImageFolder(root='dataset_train', transform=composer)
17 val_data   = ImageFolder(root='dataset_test', transform=composer)
18
19 # Wrap datasets in DataLoaders for batching and shuffling:
20 # - batch_size=16 yields mini-batches of 16 images
21 # - shuffle=True randomizes training order each epoch
22 train_loader = DataLoader(train_data, batch_size=16, shuffle=True)
23 val_loader   = DataLoader(val_data,   batch_size=16) # no shuffle for validation

1 def figure2(first_images, first_labels):
2     fig, axs = plt.subplots(1, 9, figsize=(14, 4))
3     titles = ['Offline-Module', 'Diode-Multi', 'Diode', 'Shadowing', 'Cell-Multi', 'Cell',
4     'Hot-Spot', 'Cracking', 'Hot-Spot-Multi', 'Soiling', 'Vegetation',
5     'No-Anomaly']
6     for i in range(9):
7         image, label = ToPILImage()(first_images[i]), first_labels[i]

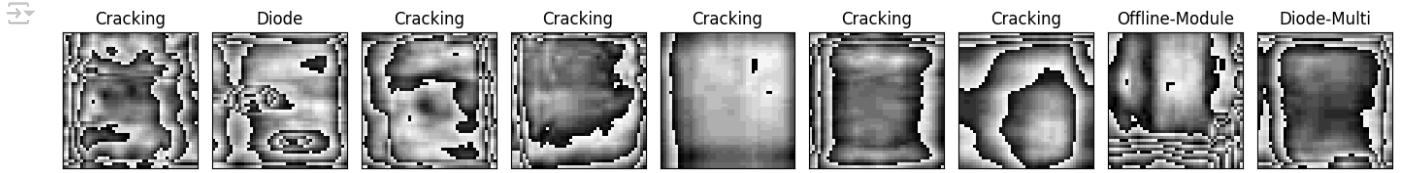
```

```

8     axs[i].imshow(image)
9     axs[i].set_xticks([])
10    axs[i].set_yticks([])
11    axs[i].set_title(titles[label], fontsize=12)
12    fig.tight_layout()
13    return fig

1 torch.manual_seed(88)
2 first_images, first_labels = next(iter(train_loader))
3
4 fig = figure2(first_images, first_labels)

```



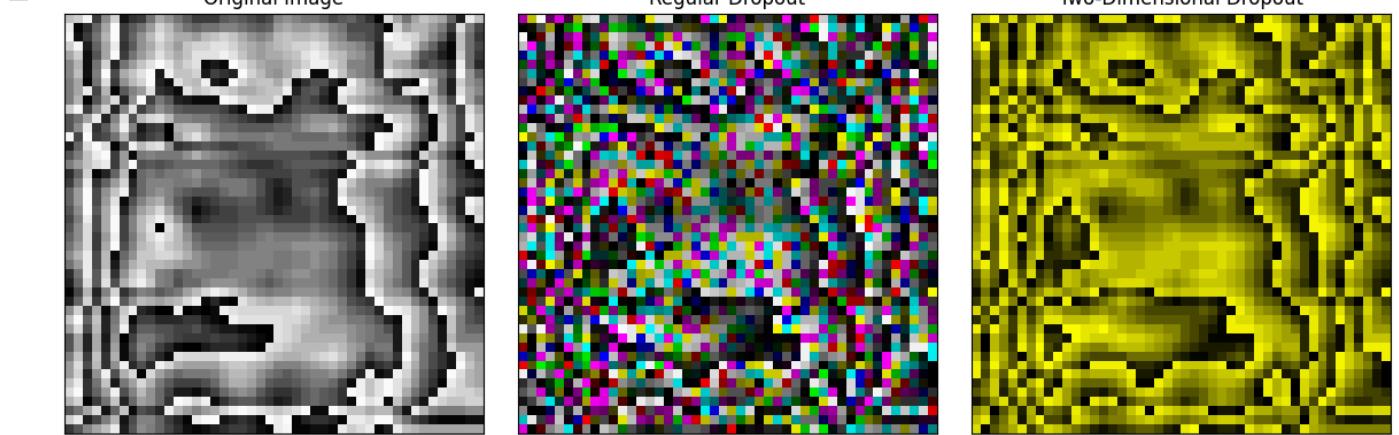
▼ Two-Dimensional Dropout

```

1 def figure9(first_images, seed=17, p=.33):
2     torch.manual_seed(seed)
3     fig, axs = plt.subplots(1, 3, figsize=(12, 4))
4     axs[0].imshow(ToPILImage()(first_images[0]))
5     axs[0].set_title('Original Image')
6     axs[0].grid(False)
7     axs[0].set_xticks([])
8     axs[0].set_yticks([])
9     axs[1].imshow(ToPILImage()(F.dropout(first_images[:1], p=p)[0]))
10    axs[1].set_title('Regular Dropout')
11    axs[1].grid(False)
12    axs[1].set_xticks([])
13    axs[1].set_yticks([])
14    axs[2].imshow(ToPILImage()(F.dropout2d(first_images[:1], p=p)[0]))
15    axs[2].set_title('Two-Dimensional Dropout')
16    axs[2].grid(False)
17    axs[2].set_xticks([])
18    axs[2].set_yticks([])
19    fig.tight_layout()
20    return fig

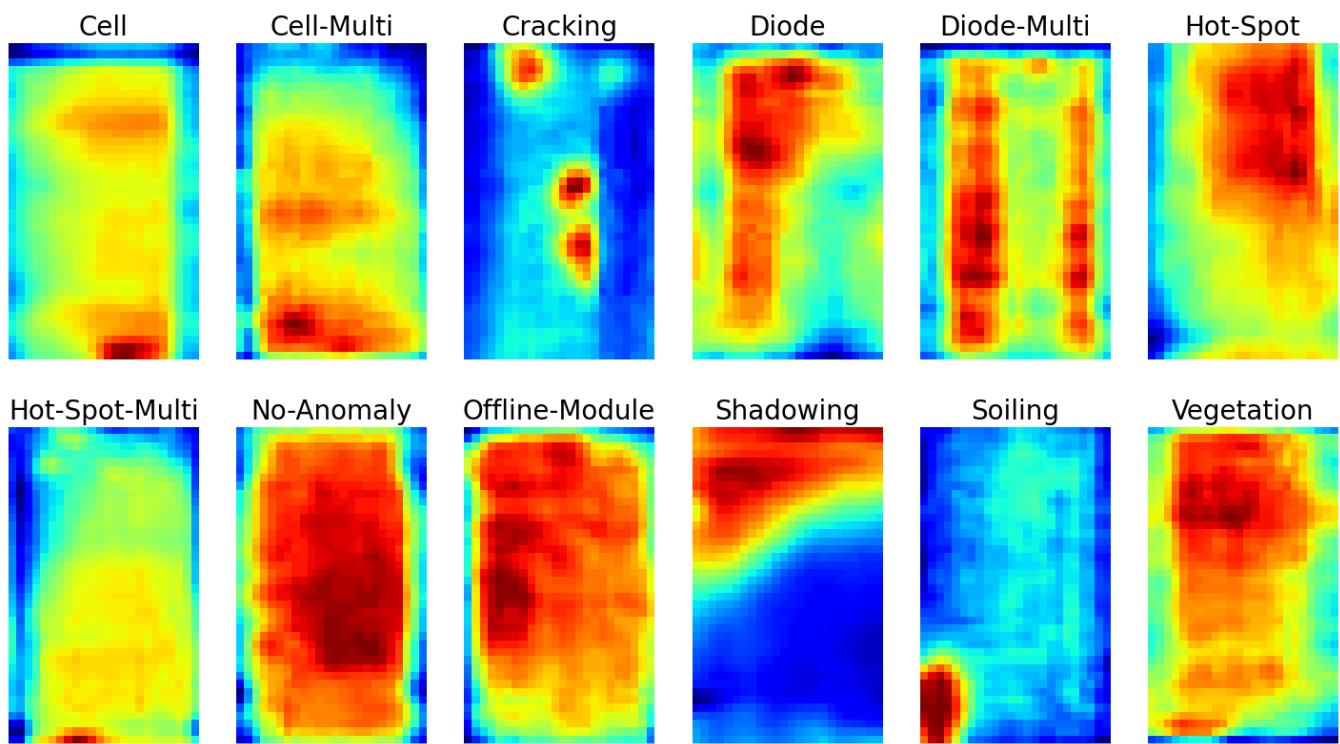
1 fig = figure9(first_images)

```



Original Dataset Image

```
1 plot()
```



▼ Modelo de ML

▼ Definindo o tipo de rede CNN2

```

1 class CNN2(nn.Module):
2     def __init__(self, n_feature, p=0.0):
3         super(CNN2, self).__init__()
4         self.n_feature = n_feature
5         self.p = p
6         # Creates the convolution layers
7         self.conv1 = nn.Conv2d(in_channels=3,
8                             out_channels=n_feature,
9                             kernel_size=3)
10        self.conv2 = nn.Conv2d(in_channels=n_feature,
11                            out_channels=n_feature,
12                            kernel_size=3)
13        # Creates the linear layers
14        # The input size of the first fully connected layer is calculated based on the output size of the last convolutional layer.
15        # For a 46x46 input image:
16        # After conv1 (kernel_size=3): (46 - 3 + 1) = 44
17        # After max_pool2d (kernel_size=2): 44 / 2 = 22
18        # After conv2 (kernel_size=3): (22 - 3 + 1) = 20
19        # After max_pool2d (kernel_size=2): 20 / 2 = 10
20        # So the output size is n_feature * 10 * 10
21        self.fc1 = nn.Linear(n_feature * 10 * 10, 50)
22        # The output layer should have 12 units, one for each anomaly class
23        self.fc2 = nn.Linear(50, 12)
24        # Creates dropout layers
25        self.drop = nn.Dropout(self.p)
26
27    def featurizer(self, x):
28        # Featurizer
29        # First convolutional block
30        # 3@46x46 -> n_feature@44x44 -> n_feature@22x22
31        x = self.conv1(x)
32        x = F.relu(x)
33        x = F.max_pool2d(x, kernel_size=2)
34        # Second convolutional block
35        # n_feature@22x22 -> n_feature@20x20 -> n_feature@10x10
36        x = self.conv2(x)
37        x = F.relu(x)
38        x = F.max_pool2d(x, kernel_size=2)
39        # Input dimension (n_feature@10x10)

```

```

40     # Output dimension (n_feature * 10 * 10)
41     x = nn.Flatten()(x)
42     return x
43
44 def classifier(self, x):
45     # Classifier
46     # Hidden Layer
47     # Input dimension (n_feature * 10 * 10)
48     # Output dimension (50)
49     if self.p > 0:
50         x = self.drop(x)
51     x = self.fc1(x)
52     x = F.relu(x)
53     # Output Layer
54     # Input dimension (50)
55     # Output dimension (12)
56     if self.p > 0:
57         x = self.drop(x)
58     x = self.fc2(x)
59     return x
60
61 def forward(self, x):
62     x = self.featurizer(x)
63     x = self.classifier(x)
64     return x

```

Model Configuration

```

1 torch.manual_seed(42)
2
3 # Model/Architecture
4 model_cnn2 = CNN2(n_feature=5, p=0.3)
5
6 # Loss function
7 multi_loss_fn = nn.CrossEntropyLoss(reduction='mean')
8
9 # Optimizer
10 optimizer_cnn2 = optim.Adam(model_cnn2.parameters(), lr=3e-4)

1 optimizer_cnn2.state_dict()

→ {'state': {}, 'param_groups': [{'lr': 0.0003, 'betas': (0.9, 0.999), 'eps': 1e-08, 'weight_decay': 0, 'amsgrad': False, 'maximize': False, 'foreach': None, 'capturable': False, 'differentiable': False, 'fused': None, 'params': [0, 1, 2, 3, 4, 5, 6, 7]}]}

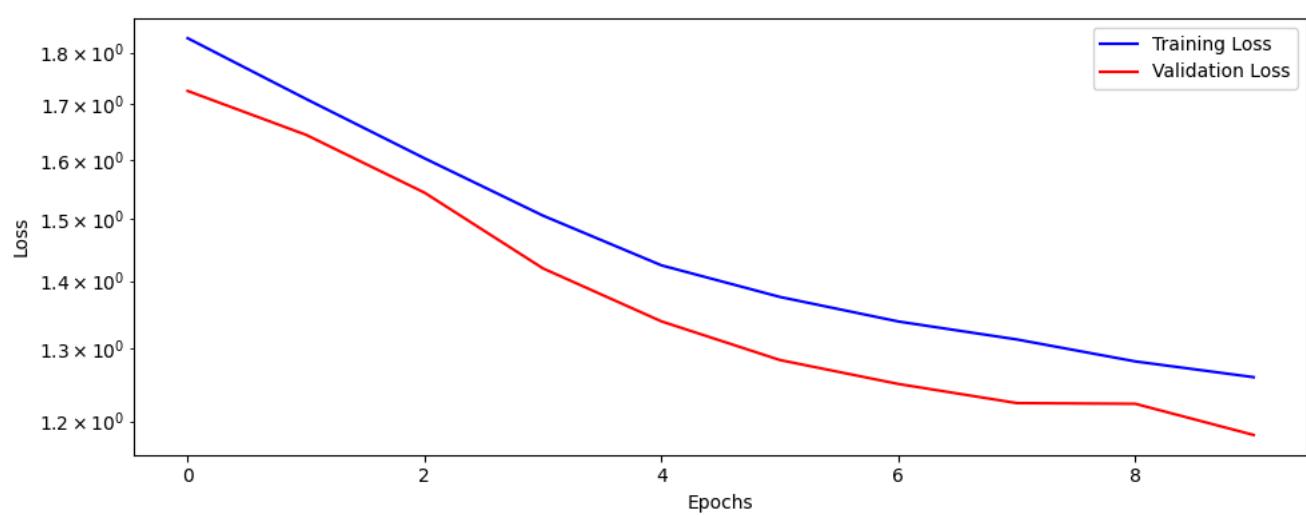
1 arch_cnn2 = Architecture(model_cnn2, multi_loss_fn, optimizer_cnn2)
2
3 arch_cnn2.set_loaders(train_loader, val_loader)
4 arch_cnn2.train(10)

1 arch_cnn2.count_parameters()

→ 26032

1 fig = arch_cnn2.plot_losses()

```



Analyzing Model with Hooks

```

1 #Extract the names of all layers, except the dropout layer.
2 todas_as_camadas = [name for name, layer in model_cnn2.named_modules() if name]
3 camadas_para_visualizar = [name for name in todas_as_camadas if 'drop' not in name]
4
5 print(f"Layers to be visualized: {camadas_para_visualizar}")
6 print("-" * 30)
7
8 try:
9     #Set the model to evaluation mode
10    arch_cnn2.model.eval()
11
12    #Attach hooks using the filtered list
13    arch_cnn2.attach_hooks(layers_to_hook=camadas_para_visualizar)
14    print("Hooks attached successfully.")
15
16    #Get a batch of images
17    torch.manual_seed(42)
18    images, labels = next(iter(val_loader))
19    images = images.to(arch_cnn2.device)
20
21    #Pass the images through the model
22    output = arch_cnn2.model(images)
23    print("Data propagated through the model.")
24
25    #Visualize the outputs
26    print("Generating visualization of feature maps...")
27    fig = arch_cnn2.visualize_outputs(layers=camadas_para_visualizar)
28    plt.show()
29    print("Visualization generated.")
30
31    #Remove the hooks
32    arch_cnn2.remove_hooks()
33    print("Hooks removed successfully.")
34
35 except Exception as e:
36     print(f"\nAn unexpected error occurred: {e}")

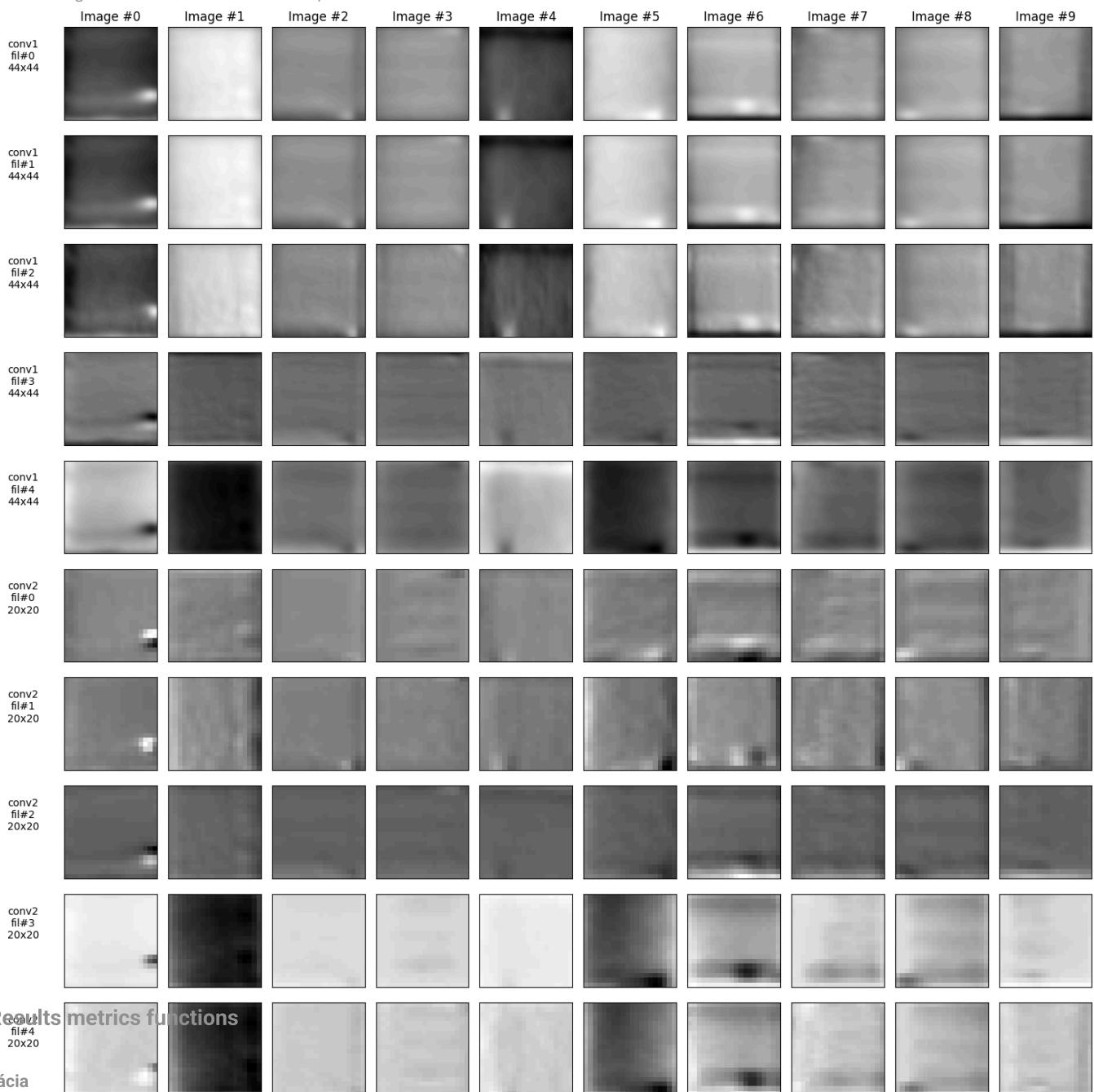
```

Layers to be visualized: ['conv1', 'conv2', 'fc1', 'fc2']

Hooks attached successfully.

Data propagated through the model.

Generating visualization of feature maps...



Results metrics functions

Acurácia

```
1 (lambda x: x[0].item() / x[1].item())(Architecture.loader_apply(val_loader,
2                                         arch_cnn2.correct).sum(axis=0))
```

Funções das métricas

fc2

```
1 # Matriz de confusão
2 def plot_confusion_matrix(arch_cnn): # Function name is confusion_matrix
3     # Get predictions and true labels for the validation set
4     all_preds = []
5     all_labels = []
6
7     arch_cnn.model.eval() # Set the model to evaluation mode
8     with torch.no_grad():
9         for inputs, labels in val_loader:
10             inputs = inputs.to(arch_cnn.device)
11             labels = labels.to(arch_cnn.device)
12
13             outputs = arch_cnn.model(inputs)
14             _, preds = torch.max(outputs, 1)
15
16             all_preds.extend(preds.cpu().numpy())
```

```

17         all_labels.extend(labels.cpu().numpy())
18
19 # Calculate the confusion matrix using the imported function
20 # This line is problematic as it calls the imported function with the same name
21 cm = confusion_matrix(all_labels, all_preds) # This will cause recursion or use the imported function
22
23 # Get class names from the validation dataset
24 class_names = val_data.classes
25
26 # Plot the confusion matrix
27 plt.figure(figsize=(10, 8))
28 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
29 plt.xlabel('Predicted Label')
30 plt.ylabel('True Label')
31 plt.title('Confusion Matrix')
32 plt.show()
33 return cm
34
35 # Precisão por classe
36 def class_precision(confusion_matrix):
37     num_classes = confusion_matrix.shape[0]
38     precision = []
39     for i in range(num_classes):
40         tp = confusion_matrix[i, i]
41         # False Positives for class i is the sum of the i-th column excluding the diagonal element
42         fp = np.sum(confusion_matrix[:, i]) - tp
43         # Handle division by zero if no predictions were made for this class
44         if tp + fp == 0:
45             precision.append(0.0) # Or np.nan, depending on preference
46         else:
47             precision.append(tp / (tp + fp))
48     return precision
49
50 # Precisão total ponderada pela dimensão das classes
51 def weighted_precision(confusion_matrix, class_counts):
52     precision_per_class = class_precision(confusion_matrix)
53     total_samples = sum(class_counts)
54     weighted_precision_sum = 0
55
56     for i, precision in enumerate(precision_per_class):
57         weight = class_counts[i] / total_samples
58         weighted_precision_sum += precision * weight
59
60     return weighted_precision_sum

```

▼ Results metrics

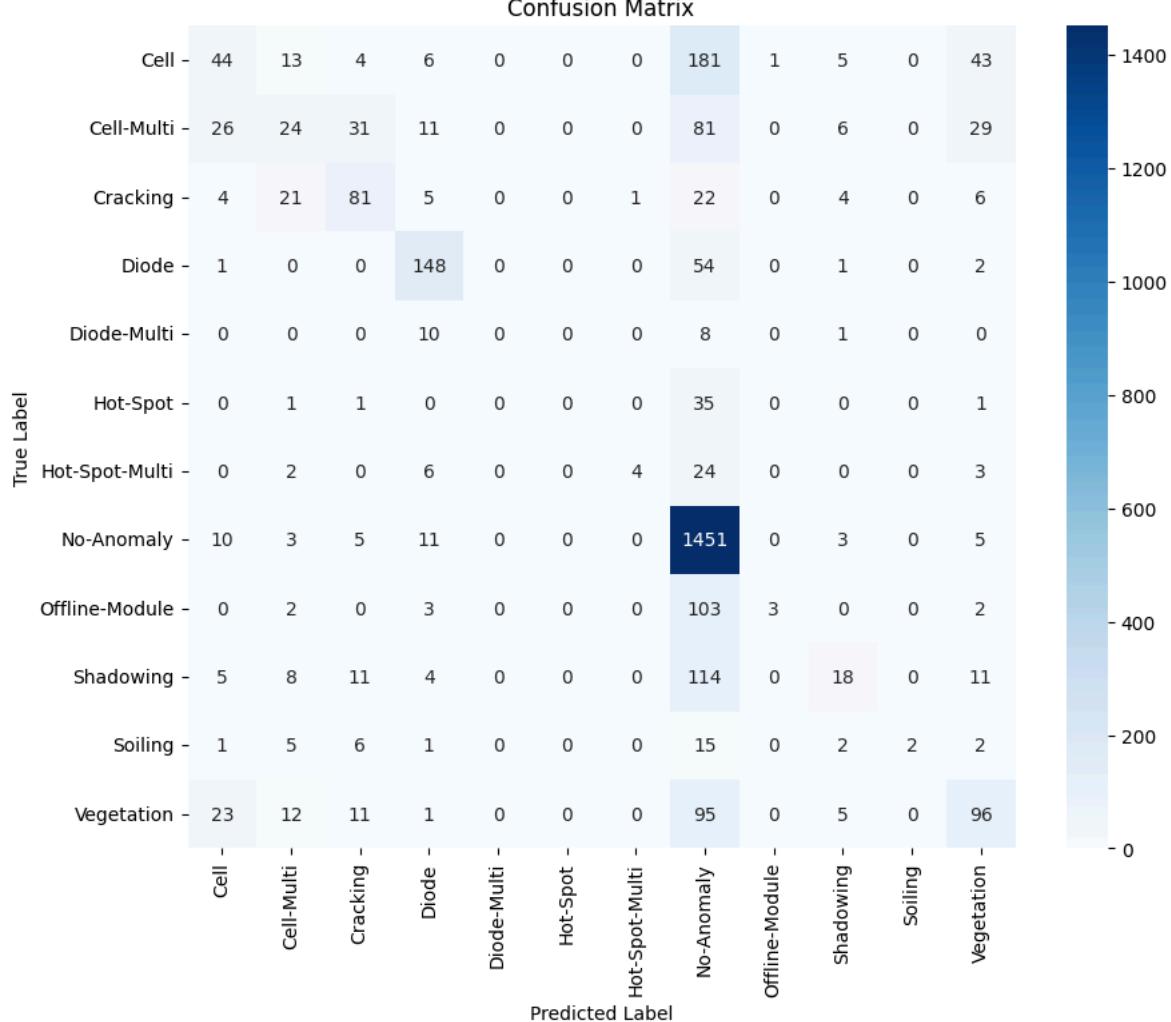
```

1 acc_train_cnn2 = (lambda x: x[0].item() / x[1].item())(Architecture.loader_apply(train_loader, arch_cnn2.correct).sum(axis=0))
2 acc_val_cnn2 = (lambda x: x[0].item() / x[1].item())(Architecture.loader_apply(val_loader, arch_cnn2.correct).sum(axis=0))
3
4 print(f"Acurácia de treinamento do modelo:{acc_train_cnn2: .5f}")
5 print(f"Acurácia de validação do modelo:{acc_val_cnn2: .5f}")

```

→ Acurácia de treinamento do modelo: 0.64818
Acurácia de validação do modelo: 0.62367

```
1 cm_arch_cnn2 = plot_confusion_matrix(arch_cnn2)
```



```

1 precision_per_class_cnn2 = class_precision(cm_arch_cnn2)
2 print("\nPrecisão por Classe (derivada da matriz de confusão):")
3 class_names = val_data.classes # Assuming val_data.classes holds the class names
4 for i, precision in enumerate(precision_per_class_cnn2):
5     print(f" Classe '{class_names[i]}': {precision:.4f}")

```

→

```

Precisão por Classe (derivada da matriz de confusão):
Classe 'Cell': 0.3860
Classe 'Cell-Multi': 0.2637
Classe 'Cracking': 0.5400
Classe 'Diode': 0.7184
Classe 'Diode-Multi': 0.0000
Classe 'Hot-Spot': 0.0000
Classe 'Hot-Spot-Multi': 0.8000
Classe 'No-Anomaly': 0.6647
Classe 'Offline-Module': 0.7500
Classe 'Shadowing': 0.4000
Classe 'Soiling': 1.0000
Classe 'Vegetation': 0.4800

```

```

1 class_counts_val = np.bincount(val_data.targets)
2 weighted_avg_precision_cnn2 = weighted_precision(cm_arch_cnn2, class_counts_val)
3 print(f"\nMédia ponderada da precisão (ponderada pela quantidade de amostras): {weighted_avg_precision_cnn2:.4f}")

```

→

```

Média ponderada da precisão (ponderada pela quantidade de amostras): 0.5731

```

Regularizing Effect

```

1 torch.manual_seed(42)
2 # Model Configuration
3 model_cnn2_nodrop = CNN2(n_feature=5, p=0.0)
4 multi_loss_fn = nn.CrossEntropyLoss(reduction='mean')
5 optimizer_cnn2_nodrop = optim.Adam(model_cnn2_nodrop.parameters(), lr=3e-4)
6 # Model Training
7 arch_cnn2_nodrop = Architecture(model_cnn2_nodrop, multi_loss_fn, optimizer_cnn2_nodrop)

```

```

8 arch_cnn2_nodrop.set_loaders(train_loader, val_loader)
9 arch_cnn2_nodrop.train(10)

1 def figure11(losses, val_losses, losses_nodrop, val_losses_nodrop):
2     fig, axs = plt.subplots(1, 1, figsize=(10, 5))
3     axs.plot(losses, 'b', label='Training Losses - Dropout')
4     axs.plot(val_losses, 'r', label='Validation Losses - Dropout')
5     axs.plot(losses_nodrop, 'b--', label='Training Losses - No Dropout')
6     axs.plot(val_losses_nodrop, 'r--', label='Validation Losses - No Dropout')
7     plt.yscale('log')
8     plt.xlabel('Epochs')
9     plt.ylabel('Loss')
10    plt.title('Regularizing Effect')
11    fig.legend(loc='lower left')
12    fig.tight_layout()
13    return fig

1 print(
2     Architecture.loader_apply(train_loader, arch_cnn2_nodrop.correct).sum(axis=0),
3     Architecture.loader_apply(val_loader, arch_cnn2_nodrop.correct).sum(axis=0)
4 )

⇒ tensor([11550, 17000]) tensor([1934, 3000])

```

```

1 print(
2     (lambda x: x[0].item() / x[1].item())(Architecture.loader_apply(train_loader, arch_cnn2_nodrop.correct).sum(axis=0)),
3     (lambda x: x[0].item() / x[1].item())(Architecture.loader_apply(val_loader, arch_cnn2_nodrop.correct).sum(axis=0))
4 )

```

⇒ 0.6794117647058824 0.6446666666666667

```

1 print(
2     Architecture.loader_apply(train_loader, arch_cnn2.correct).sum(axis=0),
3     Architecture.loader_apply(val_loader, arch_cnn2.correct).sum(axis=0)
4 )

```

⇒ tensor([11019, 17000]) tensor([1871, 3000])

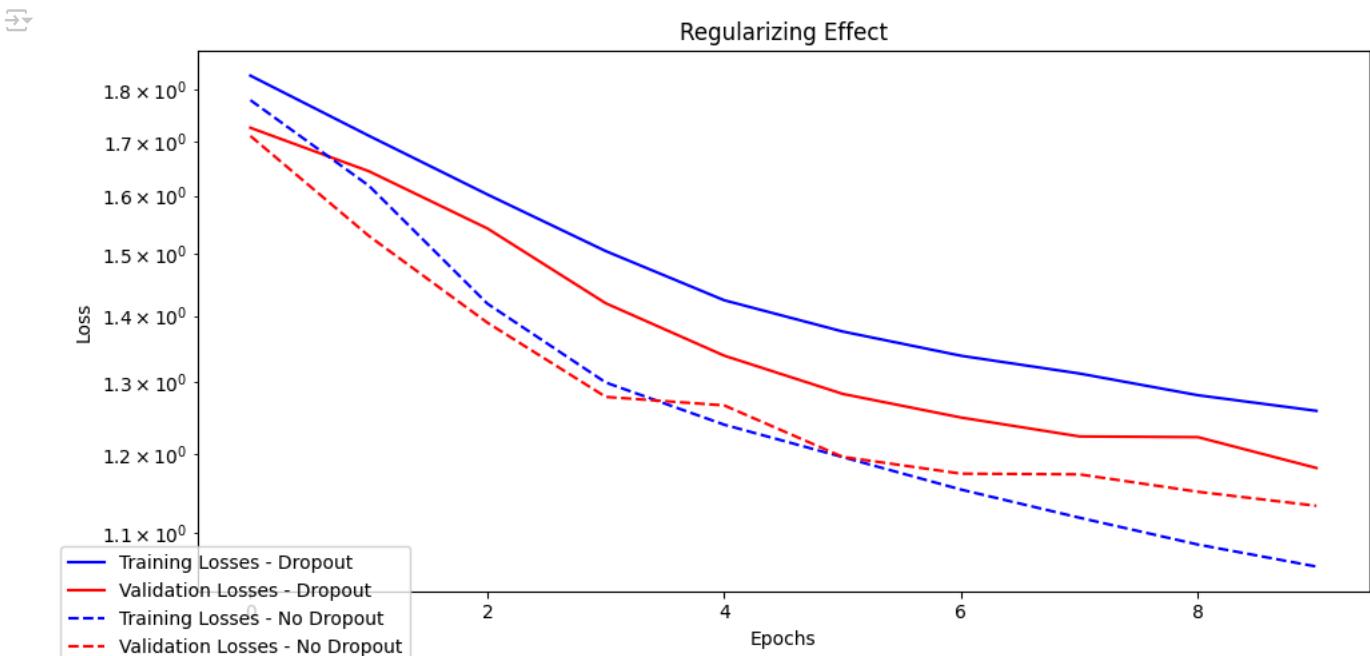
```

1 print(
2     (lambda x: x[0].item() / x[1].item())(Architecture.loader_apply(train_loader, arch_cnn2.correct).sum(axis=0)),
3     (lambda x: x[0].item() / x[1].item())(Architecture.loader_apply(val_loader, arch_cnn2.correct).sum(axis=0))
4 )

```

⇒ 0.6481764705882352 0.6236666666666667

```
1 fig = figure11(arch_cnn2.losses, arch_cnn2.val_losses, arch_cnn2_nodrop.losses, arch_cnn2_nodrop.val_losses)
```

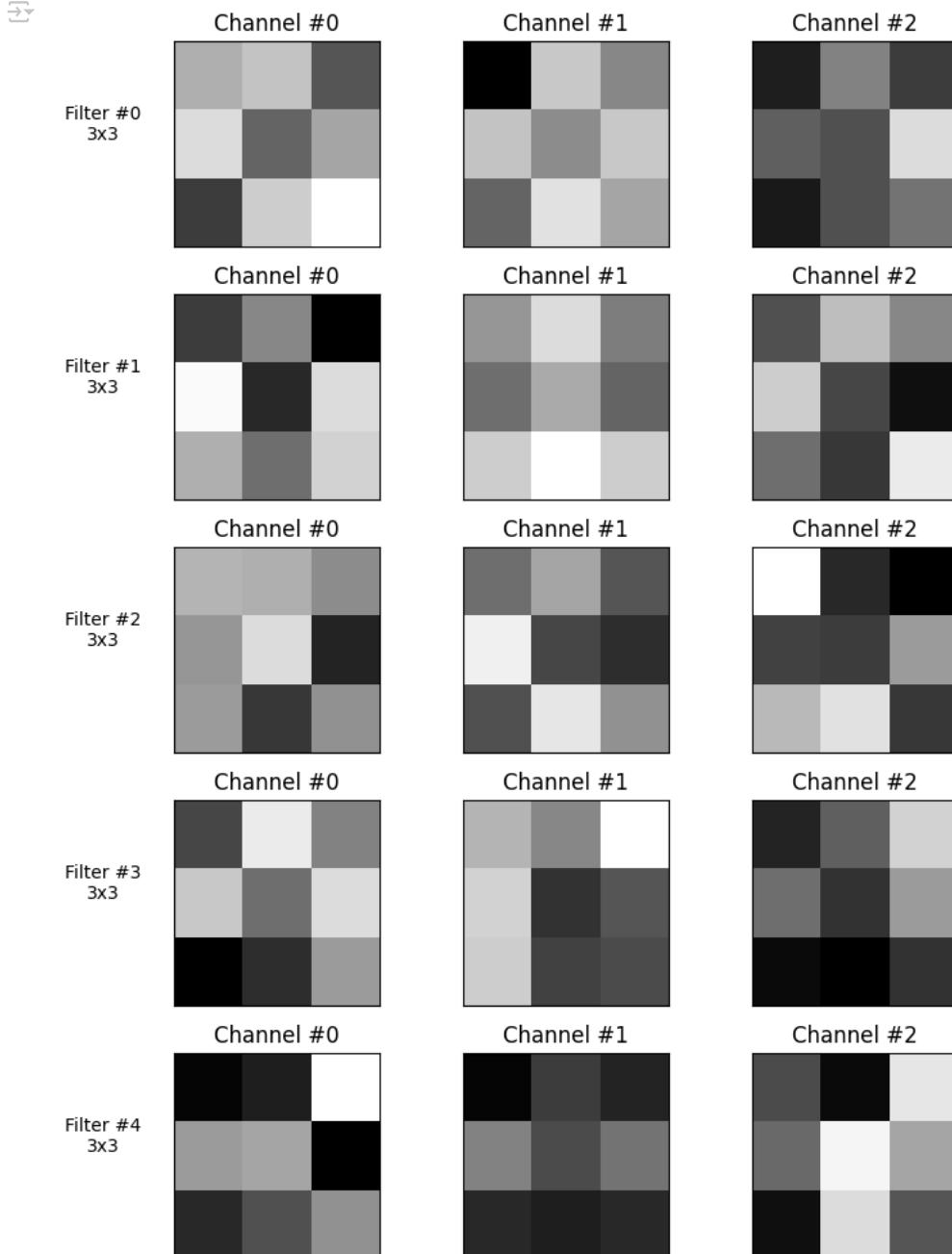


Visualizing Filters

```
1 model_cnn2.conv1.weight.shape
```

```
↳ torch.Size([5, 3, 3, 3])
```

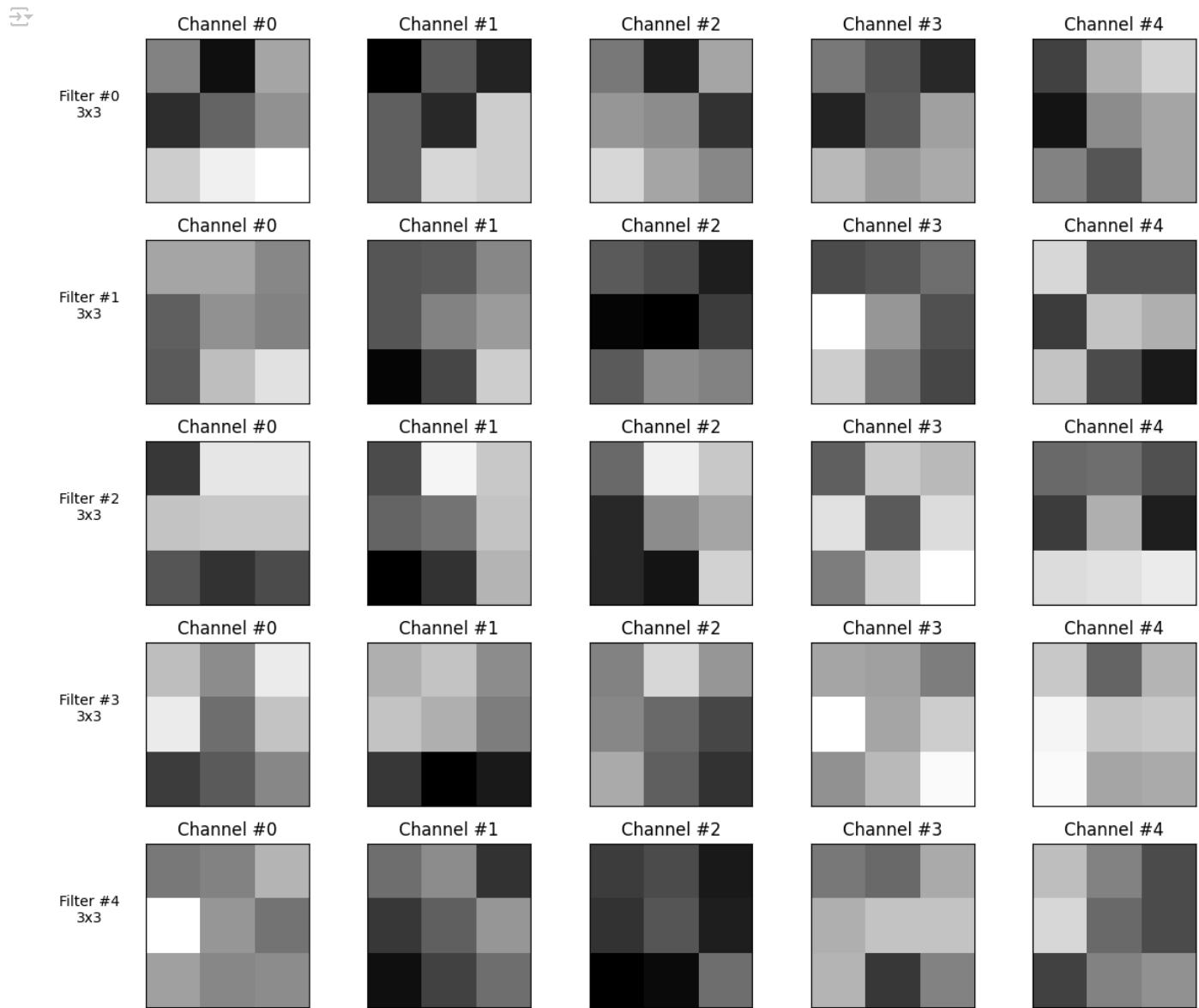
```
1 fig = arch_cnn2.visualize_filters('conv1')
```



```
1 model_cnn2.conv2.weight.shape
```

```
↳ torch.Size([5, 5, 3, 3])
```

```
1 fig = arch_cnn2.visualize_filters('conv2')
```



Confusion Matrix

```

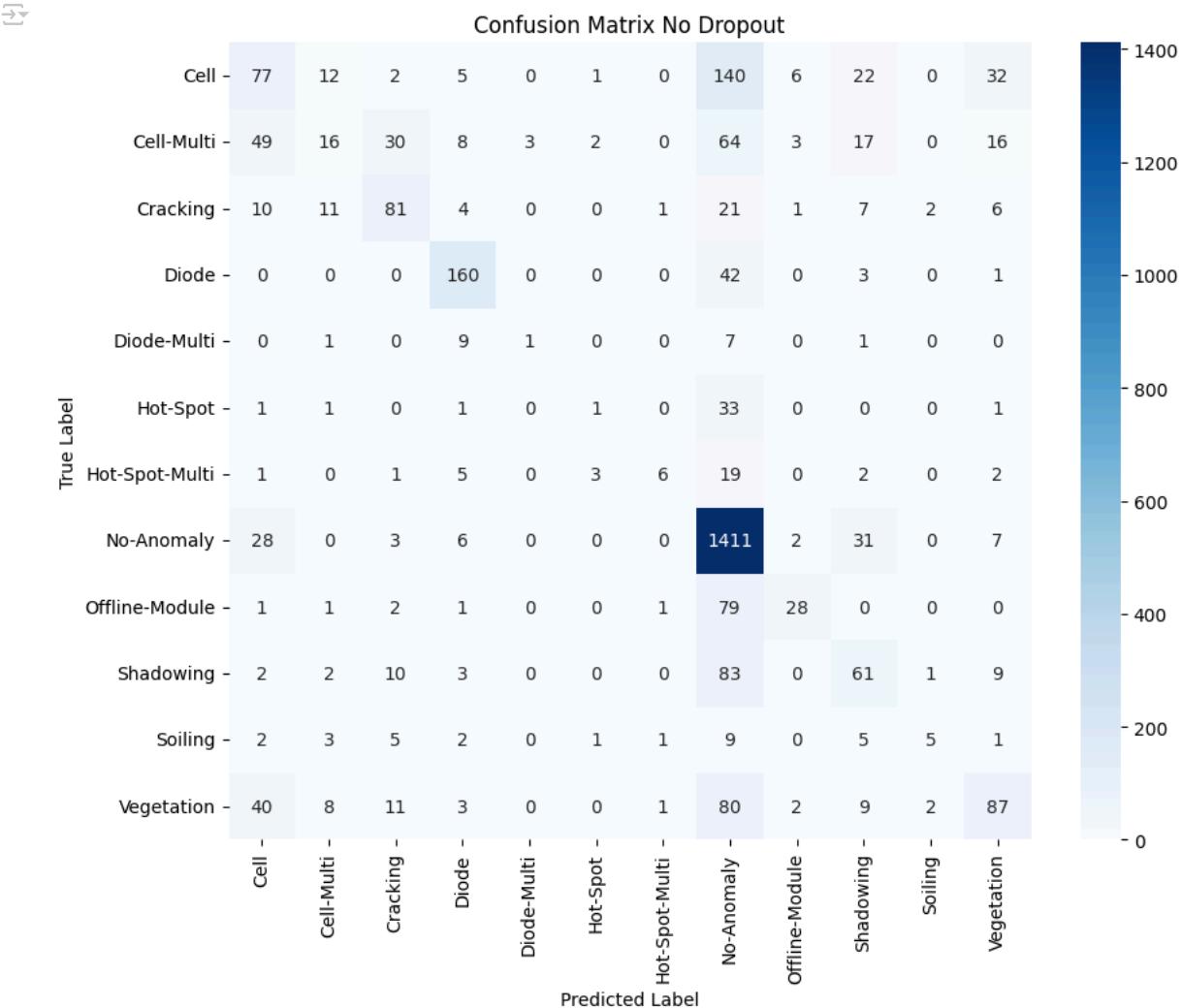
1 from sklearn.metrics import confusion_matrix
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 import numpy as np
5
6 # Get predictions and true labels for the validation set
7 all_preds = []
8 all_labels = []
9
10 arch_cnn2_nodrop.model.eval() # Set the model to evaluation mode
11 with torch.no_grad():
12     for inputs, labels in val_loader:
13         inputs = inputs.to(arch_cnn2_nodrop.device)
14         labels = labels.to(arch_cnn2_nodrop.device)
15
16         outputs = arch_cnn2_nodrop.model(inputs)
17         _, preds = torch.max(outputs, 1)
18
19         all_preds.extend(preds.cpu().numpy())
20         all_labels.extend(labels.cpu().numpy())
21
22 # Calculate the confusion matrix
23 cm = confusion_matrix(all_labels, all_preds)
24
25 # Get class names from the validation dataset
26 class_names = val_data.classes

```

```

27
28 # Plot the confusion matrix
29 plt.figure(figsize=(10, 8))
30 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
31 plt.xlabel('Predicted Label')
32 plt.ylabel('True Label')
33 plt.title('Confusion Matrix No Dropout')
34 plt.show()

```



▼ Testando diferentes modelos com variação dos hiperparâmetros

▼ Definição dos novos hiperparâmetros

```
1 !pip install optuna mlflow scikit-learn pyngrok -q
```

```

[██████████] 395.9/395.9 kB 26.5 MB/s eta 0:00:00
[██████████] 24.7/24.7 MB 68.4 MB/s eta 0:00:00
[██████████] 1.9/1.9 MB 67.6 MB/s eta 0:00:00
[██████████] 246.9/246.9 kB 18.8 MB/s eta 0:00:00
[██████████] 147.8/147.8 kB 12.4 MB/s eta 0:00:00
[██████████] 114.9/114.9 kB 9.8 MB/s eta 0:00:00
[██████████] 85.0/85.0 kB 7.0 MB/s eta 0:00:00
[██████████] 741.4/741.4 kB 46.0 MB/s eta 0:00:00
[██████████] 203.4/203.4 kB 15.2 MB/s eta 0:00:00
[██████████] 65.8/65.8 kB 5.1 MB/s eta 0:00:00
[██████████] 118.5/118.5 kB 10.0 MB/s eta 0:00:00
[██████████] 196.2/196.2 kB 17.3 MB/s eta 0:00:00

```

```

1 # Otimization
2 import optuna
3 import mlflow

```

```

1
2
3 # Helper function to get predictions

```

```

4 def get_predictions(architecture, data_loader):
5     all_preds = []
6     all_labels = []
7     architecture.model.eval()
8     with torch.no_grad():
9         for images, labels in data_loader:
10             images = images.to(architecture.device)
11             outputs = architecture.model(images)
12             _, preds = torch.max(outputs, 1)
13             all_preds.extend(preds.cpu().numpy())
14             all_labels.extend(labels.cpu().numpy())
15     return all_preds, all_labels
16
17 # Setting the name for experiment in MLflow
18 mlflow.set_experiment("Otimização do Modelo de Painéis Solares")
19
20 def objective(trial):
21     with mlflow.start_run():
22         params = {
23             'n_feature': trial.suggest_int('n_feature', 4, 16),
24             'dropout_rate': trial.suggest_float('dropout_rate', 0.1, 0.5),
25             'lr': trial.suggest_float('lr', 1e-5, 1e-3, log=True)
26         }
27         mlflow.log_params(params)
28
29         model = CNN2(n_feature=params['n_feature'], p=params['dropout_rate'])
30         optimizer = optim.Adam(model.parameters(), lr=params['lr'])
31         arch = Architecture(model, multi_loss_fn, optimizer)
32         arch.set_loaders(train_loader, val_loader)
33
34         n_epochs = 15
35         arch.train(n_epochs)
36
37         for epoch in range(n_epochs):
38             mlflow.log_metric('train_loss', arch.losses[epoch], step=epoch)
39             mlflow.log_metric('val_loss', arch.val_losses[epoch], step=epoch)
40
41         all_preds, all_labels = get_predictions(arch, val_loader)
42         class_names = val_data.classes
43         cm = confusion_matrix(all_labels, all_preds)
44         plt.figure(figsize=(12, 10))
45         sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
46         plt.title('Matriz de Confusão', fontsize=16)
47         plt.ylabel('Verdadeiro')
48         plt.xlabel('Previsto')
49         plt.xticks(rotation=45, ha='right')
50         plt.tight_layout()
51         plt.savefig("confusion_matrix.png")
52         mlflow.log_artifact("confusion_matrix.png", "plots")
53         plt.close()
54
55         final_val_loss = arch.val_losses[-1]
56         mlflow.log_metric('final_val_loss', final_val_loss)
57
58     return final_val_loss

```

```

1 study = optuna.create_study(direction='minimize')
2 study.optimize(objective, n_trials=20) # 20 models test, for exemple
3
4 print("\nEstudo de otimização finalizado!")

```

→ [I 2025-07-06 21:48:23,804] A new study created in memory with name: no-name-5981fdb2-1aba-415c-b14e-d0e920e07792
 [I 2025-07-06 21:56:42,411] Trial 0 finished with value: 1.6867249969472276 and parameters: {'n_feature': 7, 'dropout_rate': 0.16985555555555555}
 [I 2025-07-06 22:05:15,549] Trial 1 finished with value: 0.989867064309247 and parameters: {'n_feature': 8, 'dropout_rate': 0.2836511111111111}
 [I 2025-07-06 22:13:06,763] Trial 2 finished with value: 1.3875741332452347 and parameters: {'n_feature': 5, 'dropout_rate': 0.10187555555555555}
 [I 2025-07-06 22:20:57,328] Trial 3 finished with value: 1.320002841505984 and parameters: {'n_feature': 5, 'dropout_rate': 0.34248888888888885}
 [I 2025-07-06 22:29:08,718] Trial 4 finished with value: 1.0039745642942317 and parameters: {'n_feature': 7, 'dropout_rate': 0.37426666666666665}
 [I 2025-07-06 22:38:52,063] Trial 5 finished with value: 1.2208569354199348 and parameters: {'n_feature': 11, 'dropout_rate': 0.37966666666666665}
 [I 2025-07-06 22:46:49,967] Trial 6 finished with value: 1.1788716317808374 and parameters: {'n_feature': 6, 'dropout_rate': 0.45681111111111115}
 [I 2025-07-06 22:56:07,163] Trial 7 finished with value: 1.0270022054777501 and parameters: {'n_feature': 9, 'dropout_rate': 0.23448888888888885}
 [I 2025-07-06 23:03:37,889] Trial 8 finished with value: 1.271952756858866 and parameters: {'n_feature': 4, 'dropout_rate': 0.28238888888888885}
 [I 2025-07-06 23:11:43,414] Trial 9 finished with value: 1.5689510831490476 and parameters: {'n_feature': 7, 'dropout_rate': 0.38134444444444445}
 [I 2025-07-06 23:22:00,968] Trial 10 finished with value: 0.9661552502199057 and parameters: {'n_feature': 15, 'dropout_rate': 0.49444444444444445}
 [I 2025-07-06 23:32:28,013] Trial 11 finished with value: 0.9678674193614341 and parameters: {'n_feature': 16, 'dropout_rate': 0.45777777777777775}
 [I 2025-07-06 23:42:49,880] Trial 12 finished with value: 1.0090243923378752 and parameters: {'n_feature': 16, 'dropout_rate': 0.47111111111111115}
 [I 2025-07-06 23:53:13,863] Trial 13 finished with value: 1.0322127550681854 and parameters: {'n_feature': 16, 'dropout_rate': 0.49444444444444445}
 [I 2025-07-07 00:03:19,552] Trial 14 finished with value: 0.9703105480429974 and parameters: {'n_feature': 13, 'dropout_rate': 0.44111111111111115}
 [I 2025-07-07 00:13:37,464] Trial 15 finished with value: 0.9285827321099475 and parameters: {'n_feature': 14, 'dropout_rate': 0.42888888888888885}
 [I 2025-07-07 00:23:48,420] Trial 16 finished with value: 0.9667225066334644 and parameters: {'n_feature': 13, 'dropout_rate': 0.41944444444444445}
 [I 2025-07-07 00:33:55,017] Trial 17 finished with value: 0.92604703646391 and parameters: {'n_feature': 14, 'dropout_rate': 0.49911111111111115}
 [I 2025-07-07 00:43:47,911] Trial 18 finished with value: 1.1668895028055983 and parameters: {'n_feature': 12, 'dropout_rate': 0.41444444444444445}
 [I 2025-07-07 00:54:00,686] Trial 19 finished with value: 0.8891215598250323 and parameters: {'n_feature': 14, 'dropout_rate': 0.33777777777777775}

Estudo de otimização finalizado!

```

1 # Célula para iniciar a Interface do MLflow
2 from pyngrok import ngrok
3 import getpass
4
5 # Cole seu token do ngrok aqui. Ele ficará apenas nesta sessão do Colab.
6 AUTHTOKEN = ""
7 ngrok.set_auth_token(AUTHTOKEN)
8
9 #Iniciar o servidor MLflow em background (como antes)
10 get_ipython().system_raw("mlflow ui --port 5000 &")
11
12 #Criar o túnel com o ngrok
13 ngrok_tunnel = ngrok.connect(5000)
14
15 print("-----")
16 print("Interface do MLflow pronta!")
17 print("Abra este link no seu navegador para ver os resultados:")
18 print(ngrok_tunnel.public_url)
19 print("-----")

1 # Define o nome do experimento do qual queremos os dados
2 experiment_name = "Otimização do Modelo de Painéis Solares"
3
4 # Experimento pelo nome
5 try:
6     experiment = mlflow.get_experiment_by_name(experiment_name)
7     experiment_id = experiment.experiment_id
8
9     # Busca TODOS os runs (ensaios) do experimento
10    runs_df = mlflow.search_runs(experiment_ids=[experiment_id])
11
12    # Remover colunas que começam com 'params.' e 'tags.' para simplificar
13    cols_to_drop = [col for col in runs_df.columns if col.startswith('params.') or col.startswith('tags.')]
14    #runs_df_cleaned = runs_df.drop(columns=cols_to_drop)
15
16    print(f"Encontrados {len(runs_df)} resultados no experimento.")
17
18    # Salva o DataFrame completo em um arquivo CSV
19    csv_filename = "resultados_completos_otimizacao.csv"
20    runs_df.to_csv(csv_filename, index=False)
21
22    print(f"\nResultados exportados com sucesso para o arquivo: '{csv_filename}'")
23
24    # Exibe as primeiras linhas da tabela de resultados
25    print("\nAmostra dos resultados:")
26    display(runs_df.head())
27
28 except Exception as e:
29     print(f"Ocorreu um erro: {e}")
30     print("Verifique se o nome do experimento está correto e se o estudo já foi executado.")
31

```

→ Ocorreu um erro: 'NoneType' object has no attribute 'experiment_id'
Verifique se o nome do experimento está correto e se o estudo já foi executado.

```

1 #Carregar os Dados do Arquivo CSV
2
3 try:
4     # Certifique-se de que o arquivo CSV está no mesmo ambiente
5     df_results = pd.read_csv("resultados_completos_otimizacao.csv")
6     print("✅ Arquivo CSV carregado com sucesso!")
7 except FileNotFoundError:
8     print("🔴 Erro: Arquivo 'resultados_completos_otimizacao.csv' não encontrado.")
9     # Se der este erro, faça o upload do arquivo para o Colab.
10    df_results = None
11
12 if df_results is not None:
13     #Encontrar e Imprimir os Detalhes do Melhor Run ---
14
15     # Ordena o DataFrame pela menor perda de validação final
16     best_run_from_csv_val = df_results.sort_values(by="metrics.final_val_loss").iloc[0]
17     best_run_from_csv_train = df_results.sort_values(by="metrics.train_loss").iloc[0]
18
19     print("\n--- 🎯 Detalhes do Melhor Modelo (a partir do CSV) ---")
20     print(f"Loss de Validação Final: {best_run_from_csv_val['metrics.final_val_loss']:.5f}")
21     print(f"Loss de Treinamento: {best_run_from_csv_train['metrics.train_loss']:.5f}")
22
23     print("\nParâmetros do Melhor Modelo:")
24     # Filtra as colunas para pegar apenas as que começam com 'params.'

```

```

25 param_cols = [col for col in df_results.columns if col.startswith('params.')]
26 for param in param_cols:
27     param_name = param.replace('params.', '')
28     param_value = best_run_from_csv_val[param]
29     print(f" - {param_name}: {param_value}")
30
31 #Exemplo de Análise Visual Possível com o CSV
32
33 # Criar um gráfico de dispersão para ver a relação entre a taxa de aprendizado e a perda final
34 plt.figure(figsize=(10, 5))
35 sns.scatterplot(
36     data=df_results,
37     x="params.lr",
38     y="metrics.final_val_loss",
39     hue="params.n_feature", # Cor para o número de features
40     size="params.dropout_rate", # Tamanho para a taxa de dropout
41     sizes=(50, 250),
42     palette='viridis'
43 )
44 plt.title("Relação entre Taxa de Aprendizagem e Perda Final")
45 plt.xlabel("Taxa de Aprendizagem (escala log)")
46 plt.ylabel("Loss de Validação Final")
47 plt.xscale('log')
48 plt.grid(True, which='both', linestyle='--')
49 plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
50 plt.tight_layout()
51 plt.show()
52

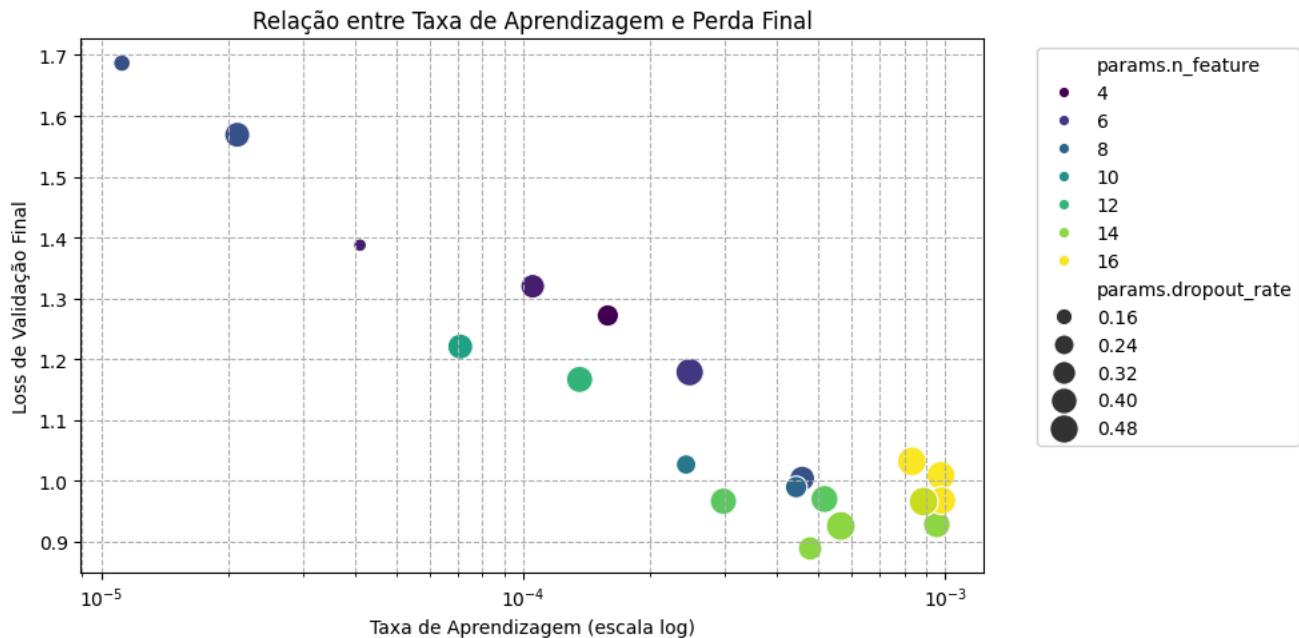
```

→ Arquivo CSV carregado com sucesso!

--- 🏆 Detalhes do Melhor Modelo (a partir do CSV) ---
 Loss de Validação Final: 0.88912
 Loss de Treinamento: 0.93424

Parâmetros do Melhor Modelo:

- lr: 0.0004788645100667
- dropout_rate: 0.3371287734951104
- n_feature: 14



```

1 #Carregar os Dados do Arquivo CSV
2
3 try:
4     # Certifique-se de que o arquivo CSV está no mesmo ambiente
5     df_results = pd.read_csv("resultados_completos_otimizacao.csv")
6     print("✅ Arquivo CSV carregado com sucesso!")
7 except FileNotFoundError:
8     print("❌ Erro: Arquivo 'resultados_completos_otimizacao.csv' não encontrado.")
9     # Se der este erro, faça o upload do arquivo para o Colab.
10    df_results = None
11
12 # Exibe as primeiras linhas da tabela de resultados
13 print("\nAmostra dos resultados:")
14 display(df_results)

```

Arquivo CSV carregado com sucesso!

Amostra dos resultados:

	run_id	experiment_id	status	artifact_uri	start_time
0	7ce379465a6144b1a14efd097a7d3da5	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/7ce3...	2025-07-07 00:43:47.916000+00:00
1	0c9a0510fe82430ba1c128848f665823	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/0c9a...	2025-07-07 00:33:55.020000+00:00
2	c5d5c6e14f254c949f63bf21385c795e	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/c5d5...	2025-07-07 00:23:48.426000+00:00
3	76a8d8c24fdd47e39612c149bb7565f0	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/76a8...	2025-07-07 00:13:37.468000+00:00
4	bdb6e2403e1f4a92becbd847a1d526d0	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/bdb6...	2025-07-07 00:03:19.557000+00:00
5	106fc20db31141abaad809efed48597e	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/106f...	2025-07-06 23:53:13.869000+00:00
6	463cf883dbb64a41a1b2a3f683d031b6	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/463c...	2025-07-06 23:42:49.885000+00:00
7	d5446148350a4ffa9abfd8d4a216c251	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/d544...	2025-07-06 23:32:28.020000+00:00
8	1280f0a12fc644058545eac74463e2a2	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/1280...	2025-07-06 23:22:00.972000+00:00
9	0d17a1e4925e4be7a6b91bb60ff0e66	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/0d17...	2025-07-06 23:11:43.418000+00:00
10	c6b51141424f469e9b9f01f792e999f8	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/c6b5...	2025-07-06 23:03:37.893000+00:00
11	1e66885640ae4841a45e4ff3b19ee027	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/1e66...	2025-07-06 22:56:07.169000+00:00
12	4c32990dd1b1427fb361593eae4f912c	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/4c32...	2025-07-06 22:46:49.972000+00:00
13	1e7e554b2af24527970edcefbd0f0fa2	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/1e7e...	2025-07-06 22:38:52.068000+00:00
14	26adb641115a417b9babcd74eff6dded	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/26ad...	2025-07-06 22:29:08.722000+00:00
15	95733da7c9b842549f1d1902e911400d	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/9573...	2025-07-06 22:20:57.332000+00:00
16	a8de8d76d1e9435cb3cfc557ff8bd817	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/a8de...	2025-07-06 22:13:06.766000+00:00
17	7df9111812b949989df2a6d2e41d08f5	893298317418703024	FINISHED	file:///content/mlruns/893298317418703024/7df9...	2025-07-06 22:05:15.554000+00:00

```

1 try:
2     # Certifique-se de que o arquivo CSV está no mesmo ambiente
3     df_results = pd.read_csv("resultados_completos_otimizacao.csv")
4     print("✅ Arquivo CSV carregado com sucesso!")
5 except FileNotFoundError:
6     print("❌ Erro: Arquivo 'resultados_completos_otimizacao.csv' não encontrado.")
7     # Se der este erro, faça o upload do arquivo para o Colab.
8     df_results = None
9
10 # Se o arquivo foi carregado, continue com a análise
11 if df_results is not None:
12     # --- 2. Encontrar e Imprimir os Detalhes do Melhor Run ---
13
14     # Ordena o DataFrame pela menor perda de validação final e pega a primeira linha (que é a melhor)
15     best_run_from_csv = df_results.sort_values(by="metrics.final_val_loss").iloc[0]
16
17     print("\n--- 🏆 Detalhes do Melhor Modelo (a partir do CSV) ---")
18     print(f"Loss de Validação Final: {best_run_from_csv['metrics.final_val_loss']:.5f}")
19
20     print("\nParâmetros do Melhor Modelo:")
21     # Filtra as colunas da tabela para pegar apenas as que começam com 'params.'
22     param_cols = [col for col in df_results.columns if col.startswith('params.')]
23     for param in param_cols:
24         # Pega o nome do parâmetro (removendo o prefixo 'params.')
25         param_name = param.replace('params.', '')
26         # Pega o valor do parâmetro da linha do melhor run
27         param_value = best_run_from_csv[param]
28
29         # Imprime de forma organizada
30         if isinstance(param_value, float):

```

```

31         print(f" - {param_name}: {param_value:.6f}")
32     else:
33         print(f" - {param_name}: {param_value}")

```

→ Arquivo CSV carregado com sucesso!

--- 🏆 Detalhes do Melhor Modelo (a partir do CSV) ---
Loss de Validação Final: 0.88912

Parâmetros do Melhor Modelo:
- lr: 0.000479
- dropout_rate: 0.337129
- n_feature: 14

```

1 import matplotlib.pyplot as plt
2 import matplotlib.image as mpimg
3
4 # Encontrar o Melhor Run (Ensaio)
5
6 # Busca todos os runs do nosso experimento, ordenando pelo menor 'final_val_loss'
7 best_run = mlflow.search_runs(
8     experiment_names=["Otimização do Modelo de Painéis Solares"],
9     order_by=["metrics.final_val_loss ASC"],
10    max_results=1
11 )
12
13 if best_run.empty:
14     print("Nenhum run encontrado. Execute a otimização primeiro.")
15 else:
16     # Pega o ID do melhor run
17     best_run_id = best_run.loc[0, 'run_id']
18     print(f"🏆 Melhor Run Encontrado! ID: {best_run_id}")
19     print(f" - Loss de Validação Final: {best_run.loc[0, 'metrics.final_val_loss']:.5f}")
20
21 # Imprimir os Hiperparâmetros do Melhor Modelo
22 print("\nParâmetros do Melhor Modelo (para recriá-lo):")
23 # Filtra as colunas para pegar apenas as que começam com 'params.'
24 best_params = {col.replace('params.', ''): best_run.loc[0, col] for col in best_run.columns if col.startswith('params.')}
25 for param_name, param_value in best_params.items():
26     # Formata o valor para ser mais legível
27     if isinstance(param_value, float):
28         print(f" - {param_name}: {param_value:.6f}")
29     else:
30         print(f" - {param_name}: {param_value}")
31
32
33 # Plotar as Curvas de Perda
34
35 # Para buscar as métricas de cada época, precisamos do cliente do MLflow
36 client = mlflow.tracking.MlflowClient()
37
38 # Busca o histórico de métricas para o nosso melhor run
39 train_loss_history = client.get_metric_history(best_run_id, "train_loss")
40 val_loss_history = client.get_metric_history(best_run_id, "val_loss")
41
42 # Extrai os valores para plotagem
43 train_losses = [m.value for m in train_loss_history]
44 val_losses = [m.value for m in val_loss_history]
45
46 plt.figure(figsize=(14, 6)) # Aumentei um pouco a figura para caber tudo
47 plt.subplot(1, 2, 1)
48 plt.plot(train_losses, label='Loss de Treinamento', color='blue')
49 plt.plot(val_losses, label='Loss de Validação', color='red', linestyle='--')
50 plt.title('Curvas de Perda do Melhor Modelo', fontsize=14)
51 plt.xlabel('Épocas')
52 plt.ylabel('Loss')
53 plt.legend()
54 plt.grid(True)
55
56 # Exibir a Matriz de Confusão
57
58 # Baixa o artefato (a imagem da matriz de confusão) do melhor run
59 try:
60     # Define o caminho local para onde o artefato será baixado
61     local_path = client.download_artifacts(run_id=best_run_id, path="plots/confusion_matrix.png")
62
63     # Carrega e exibe a imagem
64     img = mpimg.imread(local_path)
65
66     plt.subplot(1, 2, 2)
67     plt.imshow(img)
68     plt.title('Matriz de Confusão do Melhor Modelo', fontsize=14)

```

```

69     plt.axis('off') # Remove os eixos x e y da imagem
70
71 except Exception as e:
72     print(f"\nNão foi possível carregar a matriz de confusão: {e}")
73
74 plt.tight_layout()
75 plt.show()

```

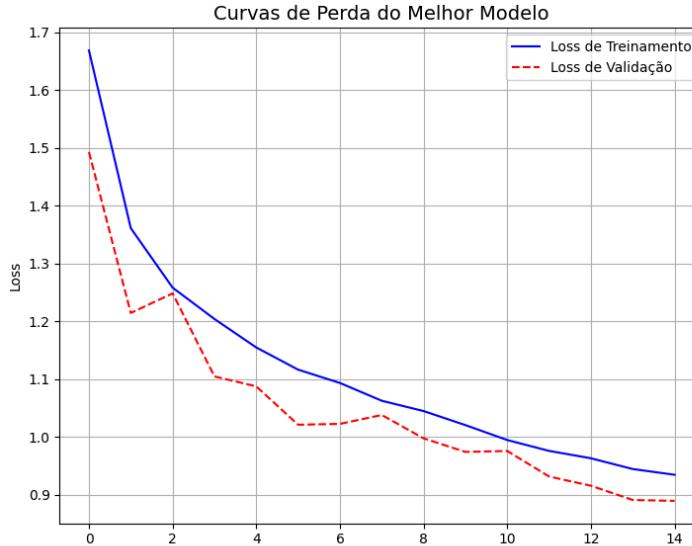
➡️ 🏆 Melhor Run Encontrado! ID: 7ce379465a6144b1a14efd097a7d3da5
- Loss de Validação Final: 0.88912

Parâmetros do Melhor Modelo (para recriá-lo):

- lr: 0.00047886451006673013
- dropout_rate: 0.3371287734951104
- n_feature: 14

Downloading artifacts: 100%

1/1 [00:00<00:00, 43.66it/s]



Matriz de Confusão do Melhor Modelo

Matriz de Confusão												
Verdadeiro	Previsto											
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing		
Cell	113	20	11	3	0	0	0	61	1	6	0	82
Cell-Multi	44	34	46	5	0	0	0	30	2	5	1	41
Cracking	13	12	96	1	0	0	2	4	0	0	1	15
Diode	4	2	0	170	0	0	0	29	0	0	0	1
Diode-Multi	0	0	0	6	5	0	0	5	2	1	0	0
Hot-Spot	3	1	2	0	0	1	0	28	0	1	0	2
Hot-Spot-Multi	1	0	1	5	1	2	8	18	0	1	0	2
No-Anomaly	8	1	1	11	0	0	1	1x54	3	3	0	6
Offline-Module	0	1	1	1	1	0	1	76	32	0	0	0
Shadowing	6	5	6	3	0	0	2	85	0	50	0	14
Solling	7	4	10	1	0	0	3	3	0	0	3	3
Vegetation	25	6	8	1	0	0	0	36	3	4	0	160

```

1 import mlflow
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import matplotlib.image as mpimg
5
6 print("---- Iniciando a Geração do Relatório Completo para Todos os Runs ---")
7
8 # Buscar os Dados de Todos os Runs ---
9
10 try:
11     # Busca TODOS os runs do experimento
12     all_runs_df = mlflow.search_runs(
13         experiment_names=["Otimização do Modelo de Painéis Solares"],
14         # Ordena para que os melhores apareçam primeiro no relatório
15         order_by=["metrics.final_val_loss ASC"]
16     )
17
18     if all_runs_df.empty:
19         print("\nNenhum run encontrado. Execute a otimização primeiro.")
20     else:
21         print(f"\nEncontrados {len(all_runs_df)} runs. Gerando relatório para cada um...")
22         # Cria o cliente do MLflow para usar no loop
23         client = mlflow.tracking.MlflowClient()
24
25         # Loop para Imprimir e Plotar Cada Run
26
27         for index, run in all_runs_df.iterrows():
28             # Pega as informações básicas do run atual
29             run_id = run['run_id']
30             final_loss = run['metrics.final_val_loss']
31
32             # Imprime um cabeçalho para separar os runs
33             print("\n" + "="*80)
34             print(f" ANALISANDO RUN #{index + 1} / {len(all_runs_df)}")
35             print(f"ID do Run: {run_id}")
36             print(f"Loss de Validação Final: {final_loss:.5f}")
37
38             # Imprime os parâmetros
39             print("\nParâmetros:")
40             param_cols = [col for col in run.index if col.startswith('params.')]
41             for param in param_cols:
42                 param_name = param.replace('params.', '')
43                 param_value = run[param]
44                 print(f" - {param_name}: {param_value}")

```

```
45
46     #Plotagem dos Gráficos
47
48     # Busca o histórico de métricas para o run atual
49     train_loss_history = client.get_metric_history(run_id, "train_loss")
50     val_loss_history = client.get_metric_history(run_id, "val_loss")
51
52     train_losses = [m.value for m in train_loss_history]
53     val_losses = [m.value for m in val_loss_history]
54
55     # Cria a figura para os gráficos
56     fig, axes = plt.subplots(1, 2, figsize=(16, 6))
57     fig.suptitle(f'Resultados do Run #{index + 1}', fontsize=16)
58
59     # Gráfico das Curvas de Perda
60     axes[0].plot(train_losses, label='Loss de Treinamento', color='blue')
61     axes[0].plot(val_losses, label='Loss de Validação', color='red', linestyle='--')
62     axes[0].set_title('Curvas de Perda', fontsize=14)
63     axes[0].set_xlabel('Épocas')
64     axes[0].set_ylabel('Loss')
65     axes[0].legend()
66     axes[0].grid(True)
67
68     # Gráfico da Matriz de Confusão
69     try:
70         local_path = client.download_artifacts(run_id=run_id, path="plots/confusion_matrix.png")
71         img = mpimg.imread(local_path)
72         axes[1].imshow(img)
73         axes[1].set_title('Matriz de Confusão', fontsize=14)
74         axes[1].axis('off')
75     except Exception as e:
76         axes[1].text(0.5, 0.5, 'Matriz de Confusão\nNão Encontrada', horizontalalignment='center', verticalalignment='center')
77         axes[1].set_title('Matriz de Confusão', fontsize=14)
78         axes[1].axis('off')
79
80     # Mostra os gráficos para o run atual antes de ir para o próximo
81     plt.show()
82
83     print("\n" + "="*80)
84     print("✅ Relatório completo gerado com sucesso!")
85
86 except Exception as e:
87     print(f"\nOcorreu um erro ao gerar o relatório: {e}")
```

--- Iniciando a Geração do Relatório Completo para Todos os Runs ---

Encontrados 20 runs. Gerando relatório para cada um...

=====
ANALISANDO RUN #1 / 20
ID do Run: 7ce379465a6144b1a14efd097a7d3da5
Loss de Validação Final: 0.88912

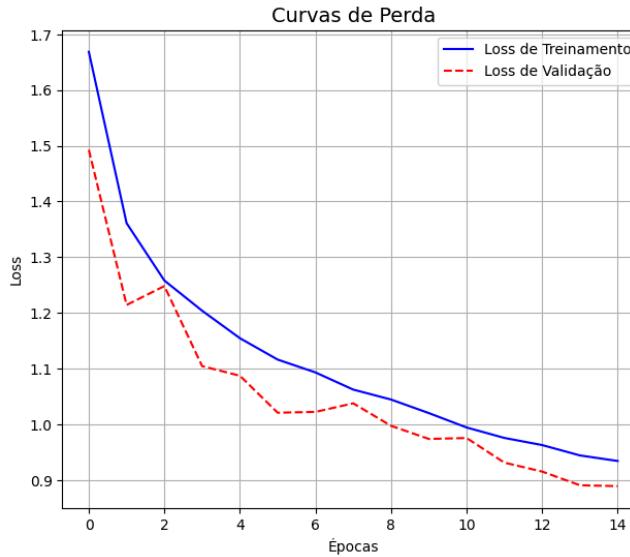
Parâmetros:

- lr: 0.00047886451006673013
- dropout_rate: 0.3371287734951104
- n_feature: 14

Downloading artifacts: 100%

1/1 [00:00<00:00, 26.34it/s]

Resultados do Run #1



Matriz de Confusão

Verdadeiro	Matriz de Confusão											
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation
Previsto	113	20	11	3	0	0	0	61	1	6	0	82
Cell	113	20	11	3	0	0	0	61	1	6	0	82
Cell-Multi	44	34	46	5	0	0	0	30	2	5	1	41
Cracking	13	12	96	1	0	0	2	4	0	0	1	35
Diode	4	2	0	170	0	0	0	29	0	0	0	1
Diode-Multi	0	0	0	6	5	0	0	5	2	1	0	0
Hot-Spot	3	1	2	0	0	1	0	28	0	1	0	2
Hot-Spot-Multi	1	0	1	5	1	2	8	18	0	1	0	2
No-Anomaly	8	1	1	11	0	0	1	1454	3	3	0	6
Offline-Module	0	1	1	1	1	0	1	76	32	0	0	0
Shadowing	6	5	6	3	0	0	2	85	0	50	0	14
Solling	7	4	10	1	0	0	3	3	0	0	3	3
Vegetation	25	6	8	1	0	0	0	36	3	4	0	160

=====
ANALISANDO RUN #2 / 20

ID do Run: c5d5c6e14f254c949f63bf21385c795e
Loss de Validação Final: 0.92605

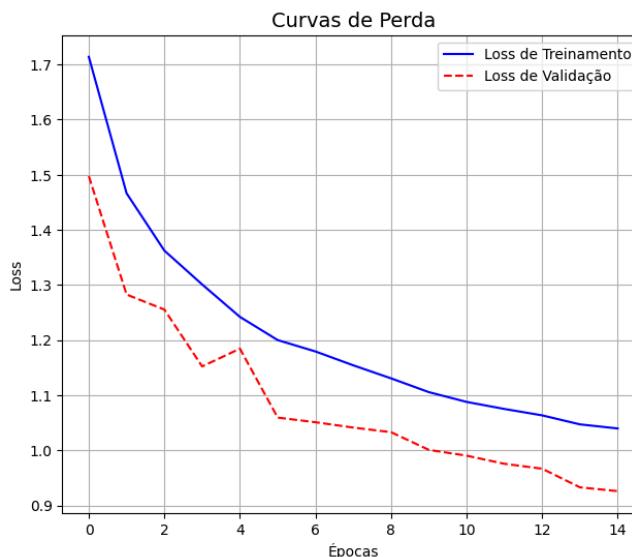
Parâmetros:

- lr: 0.0005659641838443741
- dropout_rate: 0.49911891667559455
- n_feature: 14

Downloading artifacts: 100%

1/1 [00:00<00:00, 56.91it/s]

Resultados do Run #2



Matriz de Confusão

Verdadeiro	Matriz de Confusão											
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation
Previsto	95	29	32	2	0	0	0	71	1	3	0	84
Cell	95	29	32	2	0	0	0	71	1	3	0	84
Cell-Multi	36	37	49	6	0	0	1	40	0	1	0	38
Cracking	7	16	104	0	0	0	3	5	0	0	0	9
Diode	3	3	0	172	0	0	0	28	0	0	0	0
Diode-Multi	0	1	0	6	3	0	0	9	0	0	0	0
Hot-Spot	0	1	1	0	0	0	0	1	35	0	0	0
Hot-Spot-Multi	1	0	3	4	1	0	5	23	0	0	0	2
No-Anomaly	5	5	2	9	0	0	0	0	1463	0	1	3
Offline-Module	0	1	2	0	0	0	0	90	19	0	0	1
Shadowing	14	9	7	3	0	0	1	94	0	33	0	10
Solling	4	6	12	1	0	0	2	6	0	0	0	3
Vegetation	24	4	14	0	0	0	0	54	0	2	0	145

=====
ANALISANDO RUN #3 / 20

ID do Run: bdb6e2403e1f4a92becbd847a1d526d0
Loss de Validação Final: 0.92858

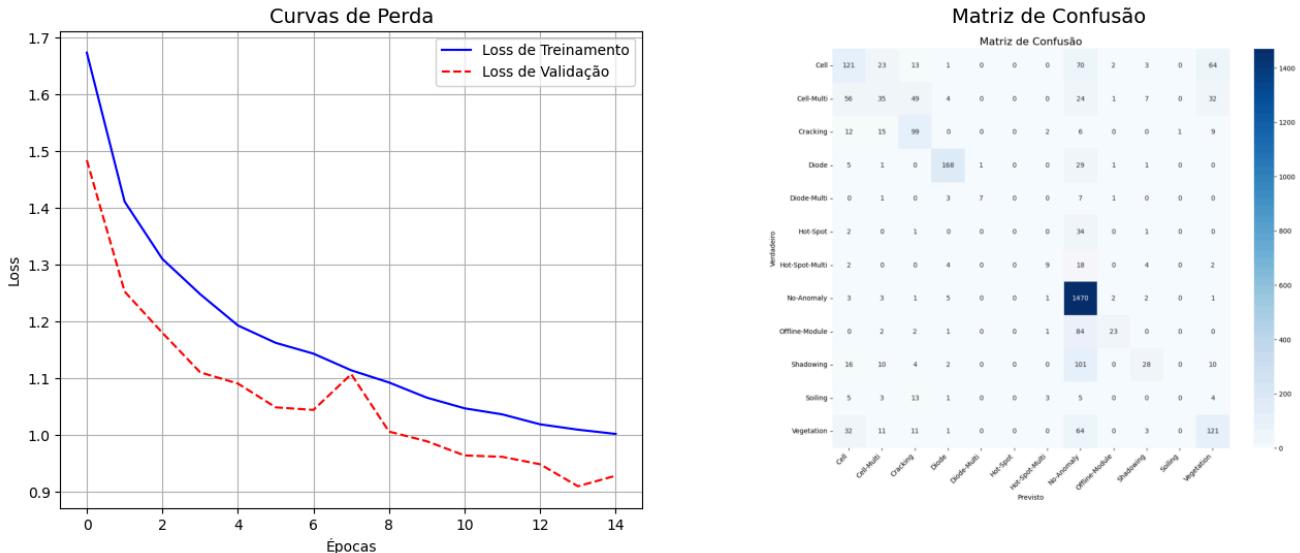
Parâmetros:

- lr: 0.0009560059228078452
- dropout_rate: 0.42824797097280914
- n_feature: 14

Downloading artifacts: 100%

1/1 [00:00<00:00, 35.25it/s]

Resultados do Run #3

**ANALISANDO RUN #4 / 20**

ID do Run: 0d17a1e4925e4be7a6b91bb60ffb0e66

Loss de Validação Final: 0.96616

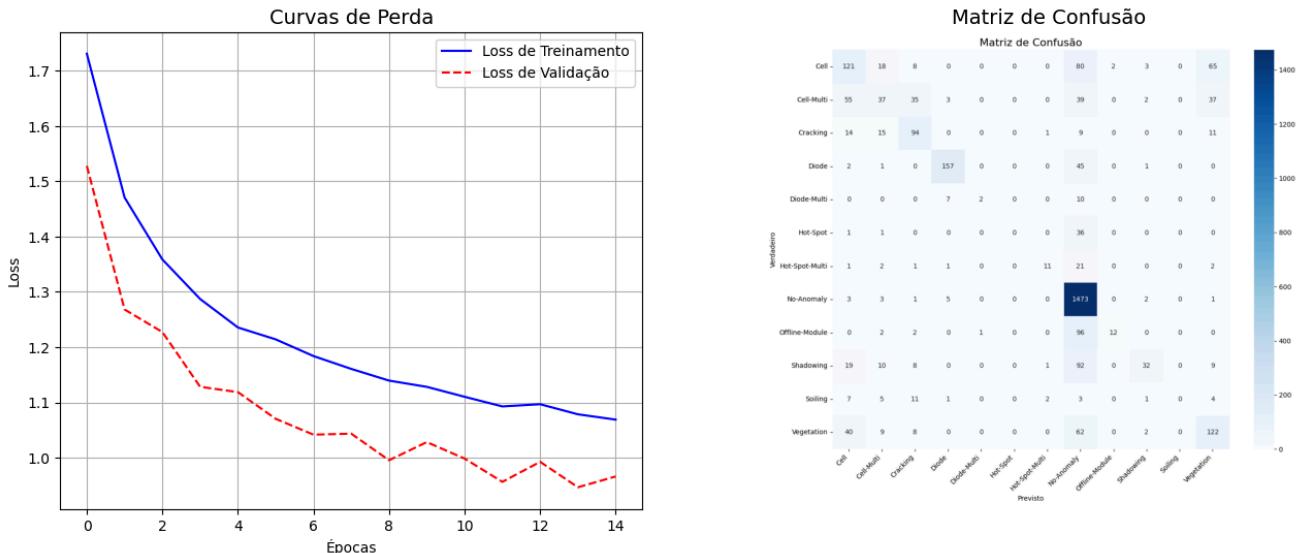
Parâmetros:

- lr: 0.0008893570067392511
- dropout_rate: 0.494771269228879
- n_feature: 15

Downloading artifacts: 100%

1/1 [00:00<00:00, 53.99it/s]

Resultados do Run #4

**ANALISANDO RUN #5 / 20**

ID do Run: 76a8d8c24fdd47e39612c149bb7565f0

Loss de Validação Final: 0.96672

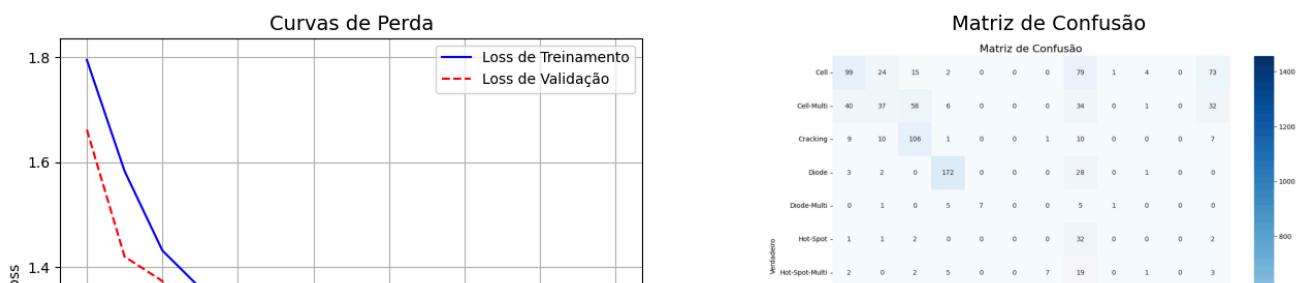
Parâmetros:

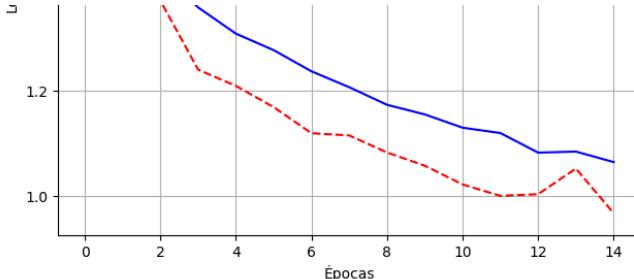
- lr: 0.0002981888093655215
- dropout_rate: 0.4192161848024607
- n_feature: 13

Downloading artifacts: 100%

1/1 [00:00<00:00, 28.02it/s]

Resultados do Run #5





	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation	
Previsto	4	6	6	7	0	0	0	0	0	0	0	0	0	0	2	5	0	2
Realizado	1456	83	97	54	0	11	341	0	0	0	0	0	0	0	24	0	0	1
Delta	-1452	-77	-91	-54	0	-11	-341	0	0	0	0	0	0	0	-22	-5	-2	-1

=====

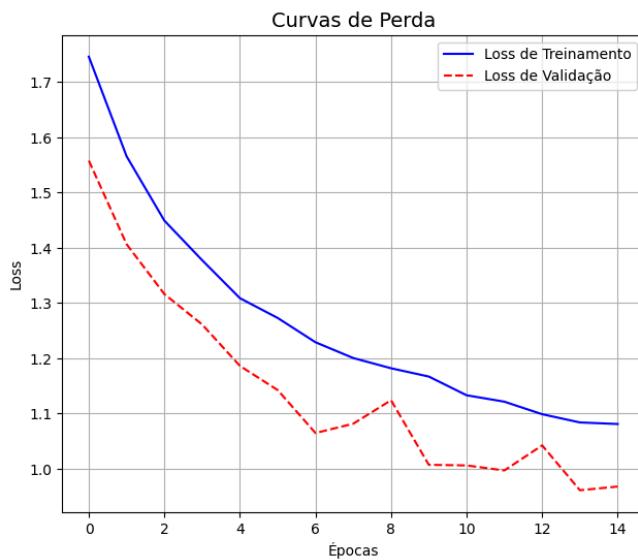
ANALISANDO RUN #6 / 20
ID do Run: 1280f0a12fc644058545eac74463e2a2
Loss de Validação Final: 0.96787

Parâmetros:
- lr: 0.0009830175263413061
- dropout_rate: 0.4570030231706655
- n_feature: 16

Downloading artifacts: 100%

1/1 [00:00<00:00, 31.48it/s]

Resultados do Run #6



Matriz de Confusão

Verdadeiro	Matriz de Confusão																		
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi
Cell	116	23	16	0	0	0	0	59	1	0	0	82	116	23	16	0	0	0	0
Cell-Multi	61	27	55	1	0	0	0	28	0	2	0	34	61	27	55	1	0	0	0
Cracking	11	14	102	0	0	0	0	7	0	0	0	10	11	14	102	0	0	0	0
Diode	3	0	0	166	0	0	0	35	1	0	0	1	3	0	0	0	0	0	0
Diode-Multi	0	1	0	7	1	0	0	7	3	0	0	0	0	1	0	0	0	0	0
Hot-Spot	2	3	0	0	0	1	0	31	0	1	0	0	2	3	0	0	0	0	0
Hot-Spot-Multi	0	3	0	5	0	0	0	22	0	0	0	4	0	3	0	0	0	0	0
No-Anomaly	10	5	0	5	0	0	0	0	91	18	0	0	10	5	0	0	0	0	0
Offline-Module	1	0	2	0	0	0	0	1	107	0	15	0	1	0	2	0	0	0	7
Shadowing	25	2	13	1	0	0	0	1	91	18	0	0	25	2	13	1	0	0	0
Solling	6	2	16	1	0	0	0	1	5	0	0	3	6	2	16	1	0	0	3
Vegetation	38	5	16	0	0	0	0	46	0	3	0	135	38	5	16	0	0	0	0

=====

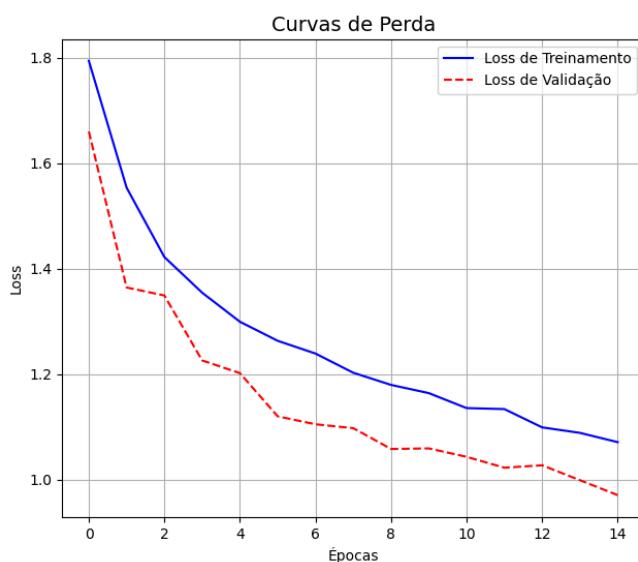
ANALISANDO RUN #7 / 20
ID do Run: 106fc20db31141abaad809efed48597e
Loss de Validação Final: 0.97031

Parâmetros:
- lr: 0.0005178437215550134
- dropout_rate: 0.44139206692138394
- n_feature: 13

Downloading artifacts: 100%

1/1 [00:00<00:00, 36.33it/s]

Resultados do Run #7



Matriz de Confusão

Verdadeiro	Matriz de Confusão																		
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi
Cell	108	24	13	2	0	0	0	82	2	3	0	63	108	24	13	2	0	0	0
Cell-Multi	34	28	58	2	0	0	0	1	44	0	2	39	34	28	58	2	0	0	0
Cracking	5	11	105	0	0	0	0	1	12	0	2	8	5	11	105	0	0	0	0
Diode	1	3	0	177	0	0	0	25	0	0	0	0	1	3	0	177	0	0	0
Diode-Multi	0	1	1	4	4	0	0	0	8	1	0	0	0	1	0	1	4	0	0
Hot-Spot	1	0	2	0	0	0	0	0	34	0	0	0	1	0	2	0	0	0	1
Hot-Spot-Multi	1	1	4	5	1	0	0	7	18	0	0	0	2	1	4	5	1	0	0
No-Anomaly	3	4	7	8	0	0	0	0	1461	2	2	1	3	4	7	8	0	0	1
Offline-Module	0	2	3	0	0	0	0	0	84	24	0	0	0	0	2	3	0	0	0
Shadowing	13	9	11	2	0	0	0	1	97	0	29	9	13	9	11	2	0	0	0
Solling	4	5	13	1	0	0	0	3	5	0	1	2	4	5	13	1	0	0	2
Vegetation	26	5	14	0	0	0	0	0	64	0	3	0	131	26	5	14	0	0	0

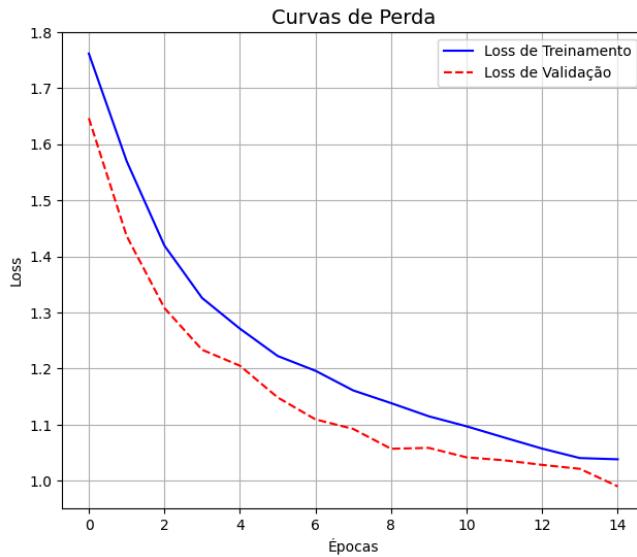
ANALISANDO RUN #8 / 20
ID do Run: 881fd02b6d0244ffbb32076eef10795
Loss de Validação Final: 0.98987

Parâmetros:
 - lr: 0.0004435327396568911
 - dropout_rate: 0.28365028413191185
 - n_feature: 8

Downloading artifacts: 100%

1/1 [00:00<00:00, 31.43it/s]

Resultados do Run #8



Matriz de Confusão

Verificado	Matriz de Confusão												Previsto
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation	
Cell	117	23	15	1	0	0	0	88	3	1	0	49	
Cell-Multi	47	51	44	4	0	0	0	37	0	0	0	24	
Cracking	11	12	102	0	0	0	1	11	0	2	0	5	
Diode	2	4	0	172	0	0	0	26	0	1	0	1	
Diode-Multi	0	2	1	5	5	0	0	6	0	0	0	0	
Hot-Spot	4	1	3	0	0	0	0	30	0	0	0	0	
Hot-Spot-Multi	2	1	2	6	1	3	4	18	0	0	0	2	
No-Anomaly	17	2	1	9	0	1	0	1451	2	2	0	3	
Offline-Module	1	0	3	0	0	0	0	84	25	0	0	0	
Shadowing	13	3	12	2	0	0	0	90	0	37	0	12	
Solling	6	6	10	2	0	0	1	5	0	2	1	1	
Vegetation	42	8	14	1	0	0	0	64	0	3	0	111	

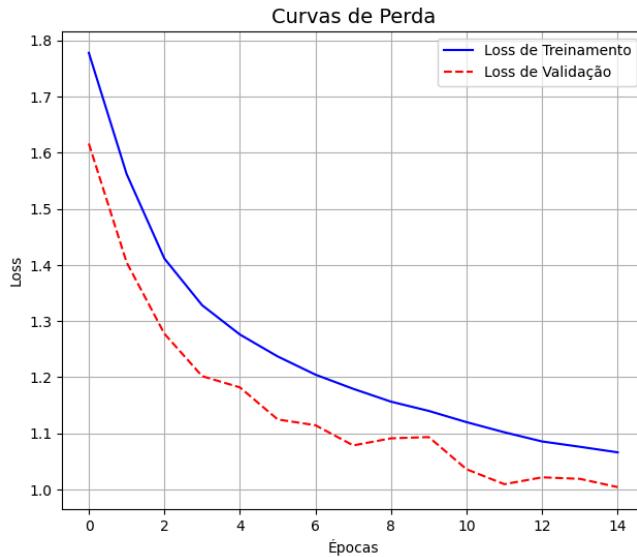
ANALISANDO RUN #9 / 20
ID do Run: 95733da7c9b842549f1d1902e911400d
Loss de Validação Final: 1.00397

Parâmetros:
 - lr: 0.0004584419071211875
 - dropout_rate: 0.37420053111721596
 - n_feature: 7

Downloading artifacts: 100%

1/1 [00:00<00:00, 34.92it/s]

Resultados do Run #9



Matriz de Confusão

Verificado	Matriz de Confusão												Previsto
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation	
Cell	115	22	8	3	0	0	1	78	1	5	0	64	
Cell-Multi	42	39	43	4	0	0	2	37	0	3	0	38	
Cracking	10	12	100	1	0	0	1	12	0	1	0	7	
Diode	5	2	2	166	0	0	0	31	0	0	0	0	
Diode-Multi	0	3	0	4	4	0	0	8	0	0	0	0	
Hot-Spot	2	5	3	1	0	0	0	1	26	0	0	0	
Hot-Spot-Multi	1	2	3	4	0	1	11	16	0	0	0	1	
No-Anomaly	8	3	5	12	0	0	0	1	1457	0	2	0	
Offline-Module	1	1	2	0	0	0	0	101	8	0	0	0	
Shadowing	15	2	12	1	0	0	0	3	110	0	17	1	10
Solling	5	5	12	1	0	0	0	1	5	0	1	2	
Vegetation	39	10	16	0	0	0	2	66	0	2	0	108	

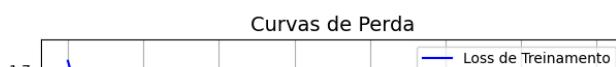
ANALISANDO RUN #10 / 20
ID do Run: d5446148350a4ffa9abfd8d4a216c251
Loss de Validação Final: 1.00902

Parâmetros:
 - lr: 0.0009776281491578342
 - dropout_rate: 0.4735942343384408
 - n_feature: 16

Downloading artifacts: 100%

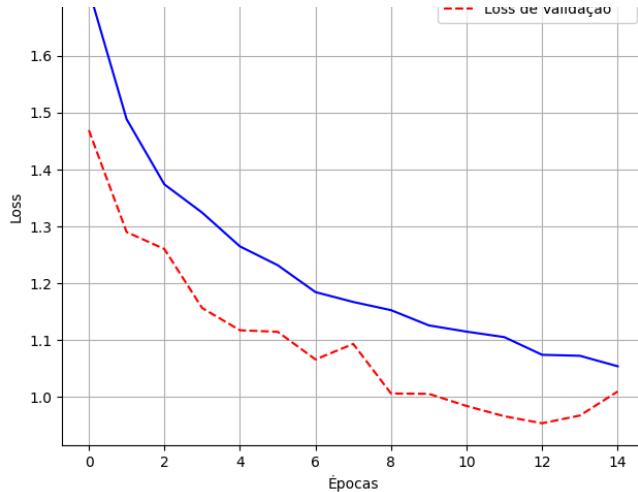
1/1 [00:00<00:00, 53.77it/s]

Resultados do Run #10



Matriz de Confusão

Verificado	Matriz de Confusão												Previsto
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation	
Cell	109	37	18	1	0	0	0	68	0	1	0	63	



	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation	Previsto
Cell	52	35	58	2	0	0	0	1	30	0	0	1	29
Cell-Multi	12	10	104	0	0	0	0	11	0	0	0	0	7
Cracking	3	1	2	153	0	0	0	47	0	0	0	0	0
Diode	0	1	0	6	2	0	0	10	0	0	0	0	0
Diode-Multi	2	4	1	0	0	0	0	31	0	0	0	0	0
Hot-Spot	2	1	6	1	0	0	0	6	21	0	0	0	2
Hot-Spot-Multi	7	7	4	9	0	0	0	3	1454	1	1	0	2
No-Anomaly	3	1	3	1	0	0	0	1	91	13	0	0	0
Offline-Module	14	12	14	3	0	0	0	3	105	0	12	0	8
Shadowing	5	2	14	1	0	0	0	2	6	0	1	1	2
Solling	37	10	22	0	0	0	1	53	1	1	0	0	118
Vegetation	0	0	0	0	0	0	0	0	0	0	0	0	0

=====

ANALISANDO RUN #11 / 20

ID do Run: 4c32990dd1b1427fb361593eae4f912c

Loss de Validação Final: 1.02700

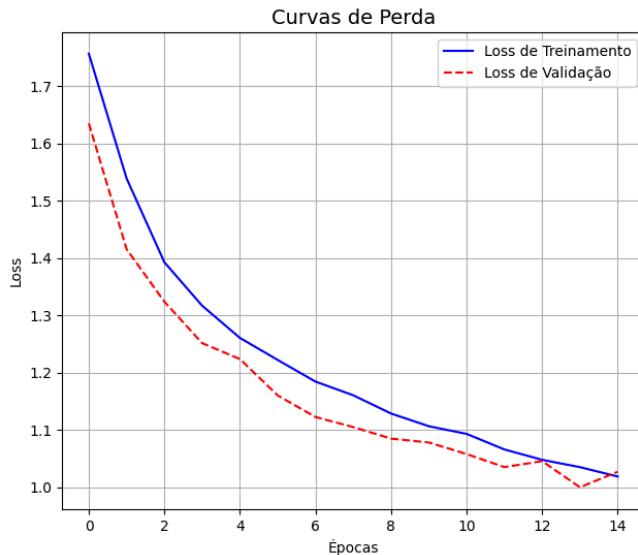
Parâmetros:

- lr: 0.0002432400716842307
- dropout_rate: 0.23448094067692607
- n_feature: 9

Downloading artifacts: 100%

1/1 [00:00<00:00, 25.03it/s]

Resultados do Run #11



Matriz de Confusão

	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation	Previsto
Cell	90	37	8	4	0	0	0	79	3	5	0	71	
Cell-Multi	27	46	56	4	0	0	0	33	1	4	0	37	
Cracking	4	15	100	2	0	0	0	2	8	0	0	1	12
Diode	4	2	0	173	0	0	0	24	0	2	0	1	
Diode-Multi	0	4	2	4	5	0	0	3	0	0	0	1	
Hot-Spot	3	4	3	1	0	3	0	23	0	0	0	1	
Hot-Spot-Multi	1	1	4	2	1	4	11	12	0	0	0	3	
No-Anomaly	13	10	6	9	1	0	0	3	1430	8	5	0	3
Offline-Module	1	4	2	0	1	0	0	3	63	35	1	0	3
Shadowing	13	6	10	2	0	0	0	3	89	0	37	0	11
Solling	3	3	9	1	0	0	0	2	7	0	2	5	2
Vegetation	17	6	16	2	0	0	0	1	48	2	3	1	147

=====

ANALISANDO RUN #12 / 20

ID do Run: 463cf883dbb64a41a1b2a3f683d031b6

Loss de Validação Final: 1.03221

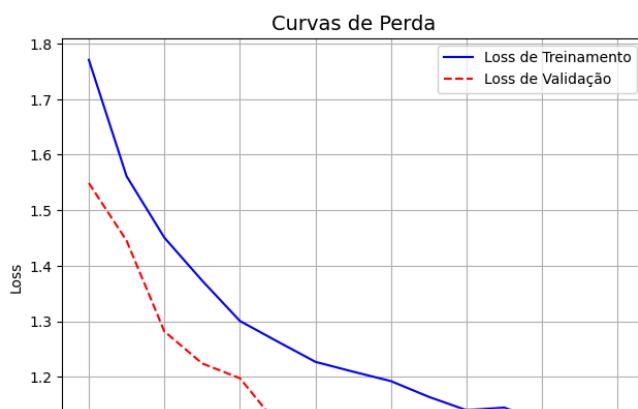
Parâmetros:

- lr: 0.0008343190339022007
- dropout_rate: 0.4940183698244794
- n_feature: 16

Downloading artifacts: 100%

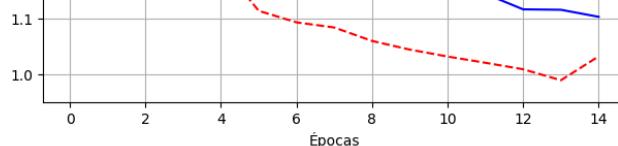
1/1 [00:00<00:00, 53.14it/s]

Resultados do Run #12



Matriz de Confusão

	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation	Previsto
Cell	94	37	18	0	0	0	0	67	0	0	0	81	
Cell-Multi	41	47	50	3	0	0	0	36	0	1	0	30	
Cracking	8	17	96	2	0	0	0	8	0	0	0	13	
Diode	0	1	0	169	0	0	0	34	0	1	0	1	
Diode-Multi	0	2	1	8	1	0	0	6	1	0	0	0	
Hot-Spot	2	2	1	0	0	0	0	33	0	0	0	0	
Hot-Spot-Multi	1	4	3	7	0	0	0	1	21	0	0	2	
No-Anomaly	9	10	0	10	0	0	0	0	1456	0	2	0	1
Offline-Module	0	4	3	1	0	0	0	0	96	9	0	0	0
Shadowing	15	15	12	2	0	0	0	0	104	0	19	0	4
Solling	2	6	15	1	0	0	0	7	0	0	0	3	



Vegetation	34	11	15	0	0	0	50	1	0	0	132
Cell	62	21	17	8	0	0	0	111	0	6	0

=====

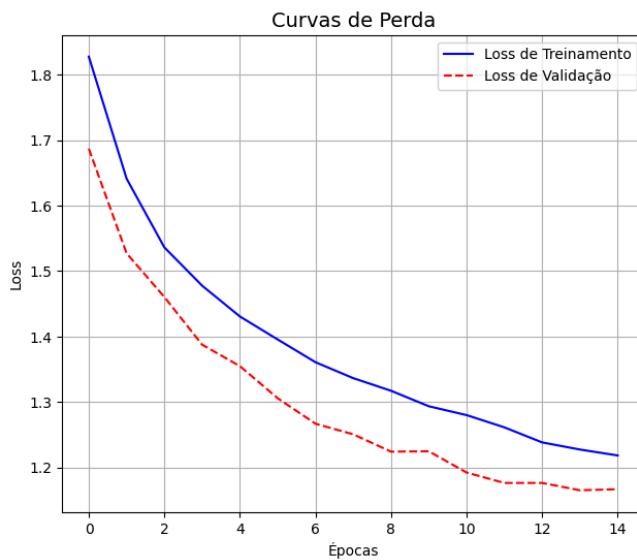
ANALISANDO RUN #13 / 20
ID do Run: 0c9a0510fe82430ba1c128848f665823
Loss de Validação Final: 1.16689

Parâmetros:
- lr: 0.00013600333114777974
- dropout_rate: 0.4147736431683601
- n_feature: 12

Downloading artifacts: 100%

1/1 [00:00<00:00, 68.95it/s]

Resultados do Run #13



Matriz de Confusão

Matriz de Confusão												
Vegetador	Previsões											
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation
Cell	62	21	17	8	0	0	0	111	0	6	0	72
Cell-Multi	30	20	66	11	0	0	0	47	0	3	0	31
Cracking	6	11	102	5	0	0	0	10	0	1	0	9
Diode	1	4	2	168	0	0	0	31	0	0	0	0
Diode-Multi	0	1	3	9	0	0	0	6	0	0	0	0
Hot-Spot	0	2	6	0	0	0	1	27	0	0	0	2
Hot-Spot-Multi	1	0	7	8	0	0	0	3	17	0	0	3
No-Anomaly	14	16	15	12	0	0	0	1	1416	0	7	0
Offline-Module	0	6	5	1	0	0	0	97	0	0	0	4
Shadowing	8	4	15	3	0	0	0	102	0	27	0	12
Solling	1	4	11	1	0	0	0	7	0	6	0	4
Vegetation	22	6	16	1	0	0	0	70	0	1	0	127

=====

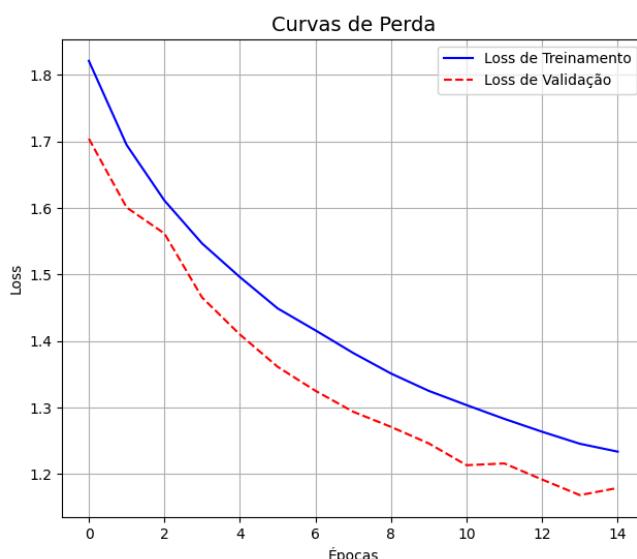
ANALISANDO RUN #14 / 20
ID do Run: 1e7e554b2af24527970edcefbd0f0fa2
Loss de Validação Final: 1.17887

Parâmetros:
- lr: 0.0002480318287915911
- dropout_rate: 0.456812612682163
- n_feature: 6

Downloading artifacts: 100%

1/1 [00:00<00:00, 45.70it/s]

Resultados do Run #14



Matriz de Confusão

Matriz de Confusão												
Vegetador	Previsões											
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation
Cell	55	25	23	5	0	0	0	117	0	0	0	72
Cell-Multi	21	19	55	7	0	0	1	60	0	1	0	44
Cracking	3	4	108	4	0	0	0	14	0	0	0	11
Diode	1	2	5	156	0	0	0	40	0	0	0	2
Diode-Multi	0	0	3	9	0	0	0	7	0	0	0	0
Hot-Spot	0	0	5	0	0	0	0	31	0	0	0	2
Hot-Spot-Multi	0	2	9	7	0	0	0	3	16	0	0	2
No-Anomaly	6	6	26	7	0	0	0	0	1437	0	1	5
Offline-Module	0	3	6	1	0	0	0	0	100	1	0	2
Shadowing	6	3	16	2	0	0	0	1	122	0	8	13
Solling	1	4	19	1	0	0	0	0	6	0	1	2
Vegetation	15	4	17	0	0	0	0	82	0	1	0	124

=====

ANALISANDO RUN #15 / 20
ID do Run: 26adb64115a417b9babcd74eff6dded
Loss de Validação Final: 1.22086

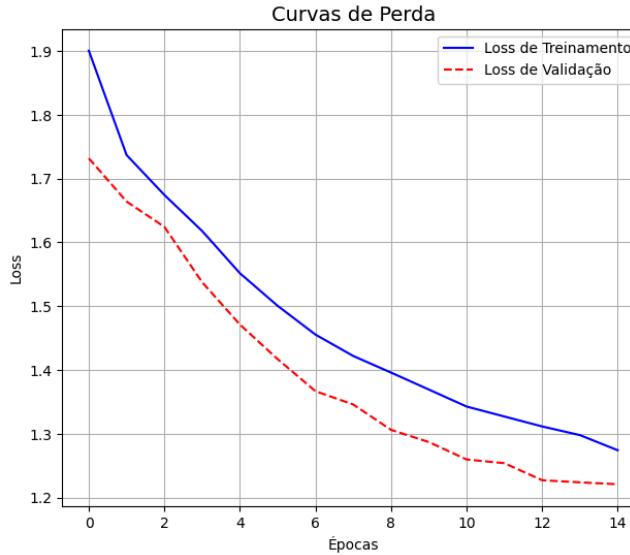
Parâmetros:
- lr: 7.089770147367083e-05

```
- dropout_rate: 0.3796657127177262
- n_feature: 11
```

Downloading artifacts: 100%

1/1 [00:00<00:00, 54.00it/s]

Resultados do Run #15



=====

ANALISANDO RUN #16 / 20

ID do Run: 1e66885640ae4841a45e4ff3b19ee027

Loss de Validação Final: 1.27195

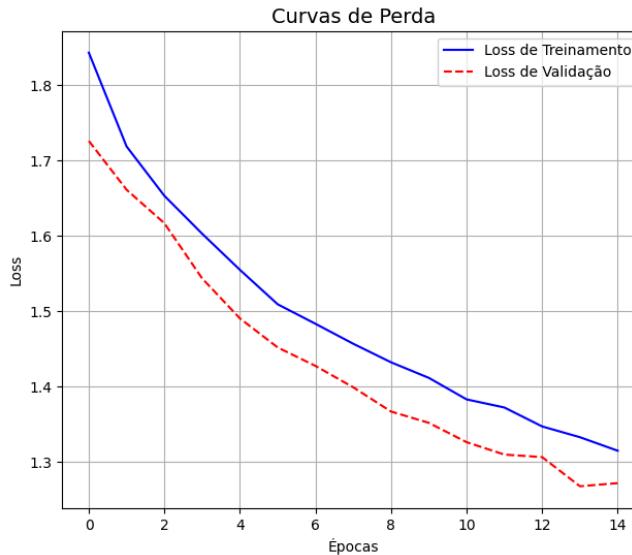
Parâmetros:

```
- lr: 0.00015865208232788787
- dropout_rate: 0.28238038153830614
- n_feature: 4
```

Downloading artifacts: 100%

1/1 [00:00<00:00, 51.02it/s]

Resultados do Run #16



=====

ANALISANDO RUN #17 / 20

ID do Run: a8de8d76d1e9435cb3cf557ff8bd817

Loss de Validação Final: 1.32000

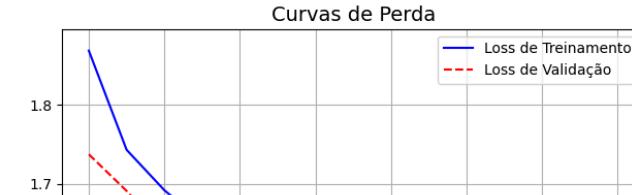
Parâmetros:

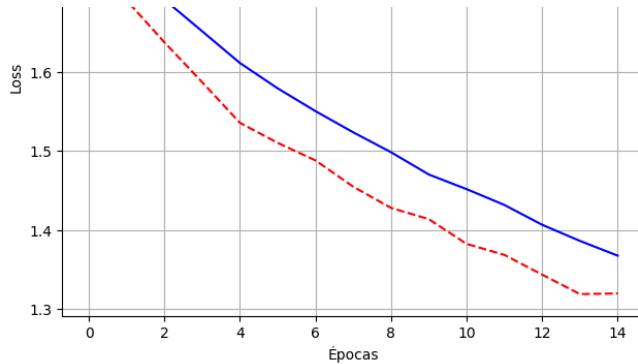
```
- lr: 0.0001052863640327261
- dropout_rate: 0.3424898657595308
- n_feature: 5
```

Downloading artifacts: 100%

1/1 [00:00<00:00, 42.40it/s]

Resultados do Run #17





Verdadeiro	Previsto											
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation
Hot-Spot	0	0	5	0	0	0	0	32	0	0	0	1
Hot-Spot-Multi	0	0	7	9	0	0	0	21	0	0	0	0
No-Anomaly	15	7	20	6	0	0	0	1430	0	1	0	9
Offline-Module	5	0	4	3	0	0	0	98	0	0	0	3
Shadowing	7	1	16	1	0	0	0	128	0	5	0	13
Solling	1	1	13	3	0	0	0	15	0	0	0	1
Vegetation	15	2	15	4	0	0	1	101	0	0	0	105

=====

ANALISANDO RUN #18 / 20
ID do Run: 7df911812b949989df2a6d2e41d08f5
Loss de Validação Final: 1.3875

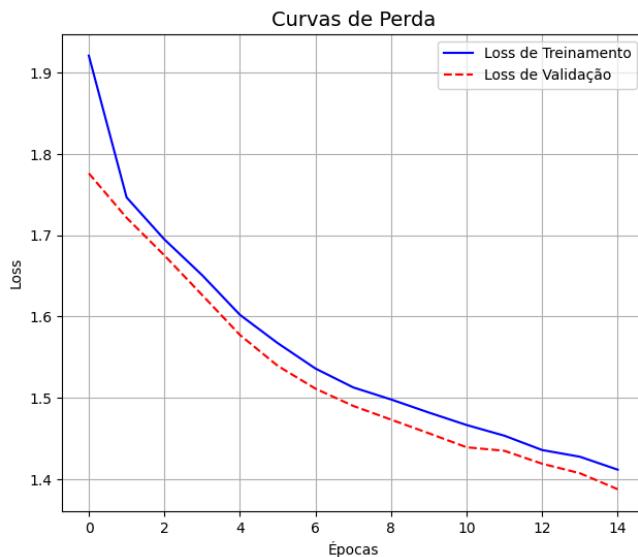
Parâmetros:

- lr: 4.102070668045529e-05
- dropout_rate: 0.10187959573942883
- n_feature: 5

Downloading artifacts: 100%

1/1 [00:00<00:00, 42.03it/s]

Resultados do Run #18



Matriz de Confusão

Verdadeiro	Previsto											
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation
Cell	22	6	2	7	0	0	0	223	0	3	0	34
Cell-Multi	12	7	26	15	0	0	1	115	1	2	0	29
Cracking	5	7	66	7	0	1	2	48	0	0	0	8
Diode	1	1	2	123	0	0	0	76	0	0	0	3
Diode-Multi	0	0	2	6	0	0	0	10	0	0	0	1
Hot-Spot	0	0	3	0	0	0	0	34	0	0	0	1
Hot-Spot-Multi	0	0	3	8	0	0	6	22	0	0	0	0
No-Anomaly	4	4	5	5	0	0	2	1461	0	1	0	6
Offline-Module	0	0	5	3	0	0	0	102	3	0	0	0
Shadowing	2	2	8	2	0	0	1	133	0	10	0	13
Solling	1	2	8	3	0	0	1	18	0	0	0	1
Vegetation	10	3	6	4	0	0	1	146	0	0	0	73

=====

ANALISANDO RUN #19 / 20
ID do Run: c6b51141424f469e9b9f01f792e999f8
Loss de Validação Final: 1.56895

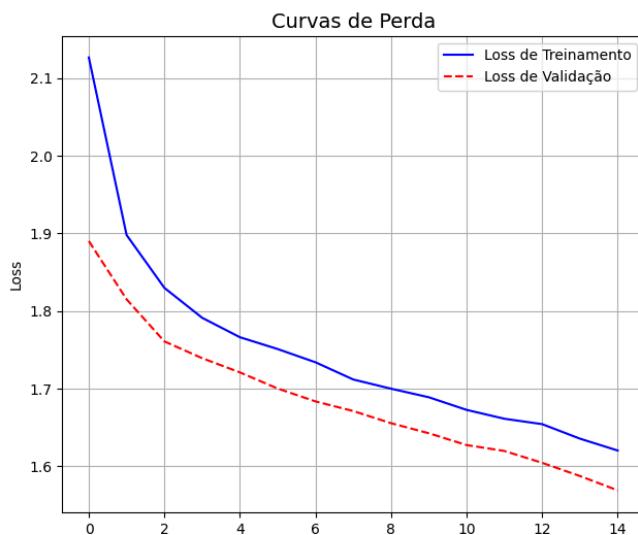
Parâmetros:

- lr: 2.098088282695299e-05
- dropout_rate: 0.38134458534086124
- n_feature: 7

Downloading artifacts: 100%

1/1 [00:00<00:00, 24.82it/s]

Resultados do Run #19



Matriz de Confusão

Verdadeiro	Previsto											
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation
Cell	10	0	0	1	0	0	0	252	0	0	0	34
Cell-Multi	10	0	0	3	0	0	0	173	0	0	0	22
Cracking	3	1	0	3	0	0	0	117	0	0	0	20
Diode	0	0	0	24	0	0	0	180	0	0	0	2
Diode-Multi	0	0	0	5	0	0	0	14	0	0	0	0
Hot-Spot	0	0	0	0	0	0	0	38	0	0	0	0
Hot-Spot-Multi	0	0	0	4	0	0	0	35	0	0	0	0
No-Anomaly	0	0	0	0	0	0	0	1477	0	0	0	11
Offline-Module	0	0	0	0	0	0	0	113	0	0	0	0
Shadowing	0	0	1	0	0	0	0	158	0	2	0	10
Solling	2	0	0	1	0	0	0	30	0	0	0	1
Vegetation	7	0	0	1	0	0	0	160	0	0	0	75

Épocas

=====

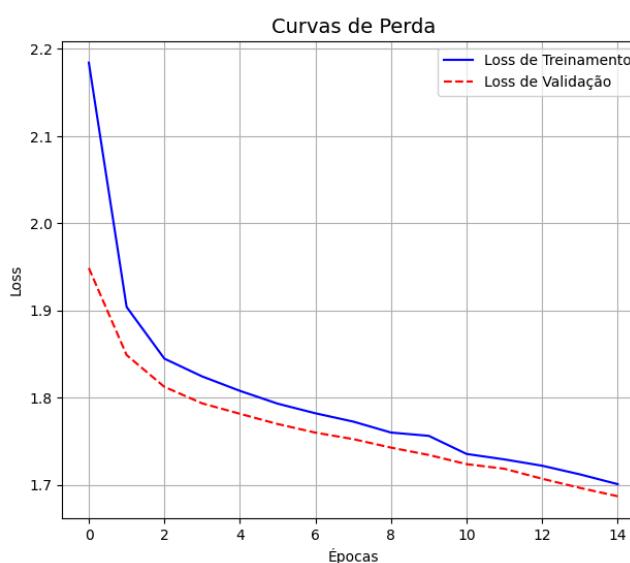
ANALISANDO RUN #20 / 20
ID do Run: 13a251cf94e24bab936c20a4f2d612a9
Loss de Validação Final: 1.68672

Parâmetros:
 - lr: 1.1177285796090666e-05
 - dropout_rate: 0.16989998387525254
 - n_feature: 7

Downloading artifacts: 100%

1/1 [00:00<00:00, 46.23it/s]

Resultados do Run #20



Matriz de Confusão

Matriz de Confusão												
Real	Previsto											
	Cell -	Cell-Multi -	Cracking -	Diode -	Diode-Multi -	Hot-Spot -	Hot-Spot-Multi -	No-Anomaly -	Offline-Module -	Shadowing -	Selling -	Vegetation -
Cell -	0	0	0	0	0	0	0	291	0	0	0	6
Cell-Multi -	0	0	0	0	0	0	0	204	0	0	0	4
Cracking -	0	0	0	0	0	0	0	143	0	0	0	1
Diode -	0	0	0	0	0	0	0	205	0	0	0	1
Diode-Multi -	0	0	0	0	0	0	0	19	0	0	0	0
Hot-Spot -	0	0	0	0	0	0	0	38	0	0	0	0
Hot-Spot-Multi -	0	0	0	0	0	0	0	38	0	0	0	1
No-Anomaly -	0	0	0	0	0	0	0	1480	0	0	0	8
Offline-Module -	0	0	0	0	0	0	0	113	0	0	0	0
Shadowing -	0	0	0	0	0	0	0	164	0	6	0	1
Selling -	0	0	0	0	0	0	0	34	0	0	0	0
Vegetation -	0	0	0	0	0	0	0	222	0	0	0	21

=====

Relatório completo gerado com sucesso!


```

1 import mlflow
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import matplotlib.image as mpimg
5
6 print(" --- Starting Generation of the Full Report for All Runs ---")
7
8 # --- 1. Fetch Data for All Runs ---
9
10 try:
11     # Fetch ALL runs from the experiment
12     all_runs_df = mlflow.search_runs(
13         experiment_names=["Otimização do Modelo de Painéis Solares"],
14         # Sort so the best models appear first in the report
15         order_by=["metrics.final_val_loss ASC"]
16     )
17
18     if all_runs_df.empty:
19         print("\nNo runs found. Please run the optimization first.")
20     else:
21         print(f"\nFound {len(all_runs_df)} runs. Generating a report for each one...")
22         # Create the MLflow client to be used inside the loop
23         client = mlflow.tracking.MlflowClient()
24
25     # --- 2. Loop to Print and Plot Each Run ---
26
27     for index, run in all_runs_df.iterrows():
28         # Get basic info for the current run
29         run_id = run['run_id']
30         final_loss = run['metrics.final_val_loss']
31
32         # Print a header to separate the runs
33         print("\n" + "="*80)
34         print(f"\u25b2 ANALYZING RUN #{index + 1} / {len(all_runs_df)}")
35         print(f"Run ID: {run_id}")
36         print(f"Final Validation Loss: {final_loss:.5f}")
37
38         # Print the parameters
39         print("\nParameters:")
40         param_cols = [col for col in run.index if col.startswith('params.')]
41         for param in param_cols:
42             param_name = param.replace('params.', '')
43             param_value = run[param]
44             print(f" - {param_name}: {param_value}")
45
46     # --- Plotting the Graphs ---
47
48     # Fetch the metric history for the current run
49     train_loss_history = client.get_metric_history(run_id, "train_loss")
50     val_loss_history = client.get_metric_history(run_id, "val_loss")
51
52     train_losses = [m.value for m in train_loss_history]
53     val_losses = [m.value for m in val_loss_history]
54
55     # Create the figure for the plots
56     fig, axes = plt.subplots(1, 2, figsize=(16, 6))
57     fig.suptitle(f'Run #{index + 1} Results', fontsize=16)
58
59     # Loss Curves Plot
60     axes[0].plot(train_losses, label='Training Loss', color='blue')
61     axes[0].plot(val_losses, label='Validation Loss', color='red', linestyle='--')
62     axes[0].set_title('Loss Curves', fontsize=14)
63     axes[0].set_xlabel('Epochs')
64     axes[0].set_ylabel('Loss')
65     axes[0].legend()
66     axes[0].grid(True)
67
68     # Confusion Matrix Plot
69     try:
70         local_path = client.download_artifacts(run_id=run_id, path="plots/confusion_matrix.png")
71         img = mpimg.imread(local_path)
72         axes[1].imshow(img)
73         axes[1].set_title('Confusion Matrix', fontsize=14)
74         axes[1].axis('off')
75     except Exception as e:
76         axes[1].text(0.5, 0.5, 'Confusion Matrix\nNot Found', horizontalalignment='center', verticalalignment='center')
77         axes[1].set_title('Confusion Matrix', fontsize=14)
78         axes[1].axis('off')
79
80     # Show the plots for the current run before moving to the next
81     plt.show()
82

```

```
83     print("\n" + "*80")
84     print("✅ Full report generated successfully!")
85
86 except Exception as e:
87     print(f"\nAn error occurred while generating the report: {e}")
```

--- Starting Generation of the Full Report for All Runs ---

Found 20 runs. Generating a report for each one...

```
=====
[1] ANALYZING RUN #1 / 20
Run ID: 7ce379465a6144b1a14efd097a7d3da5
Final Validation Loss: 0.88912
```

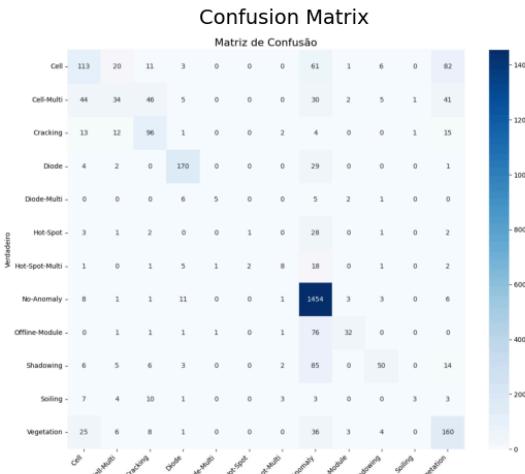
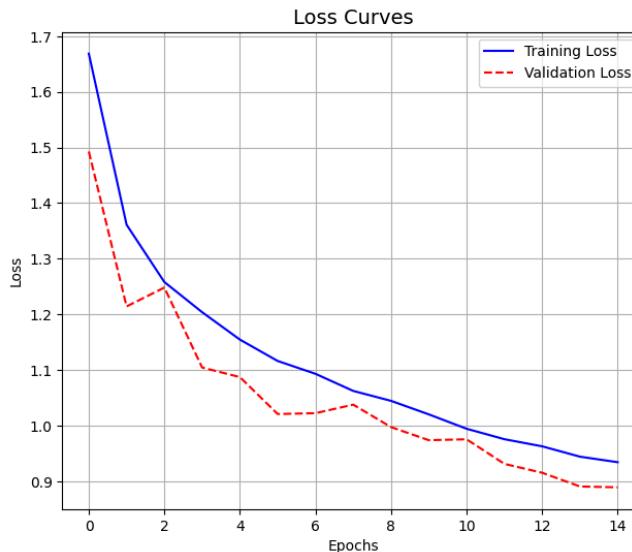
Parameters:

- lr: 0.00047886451006673013
- dropout_rate: 0.3371287734951104
- n_feature: 14

Downloading artifacts: 100%

1/1 [00:00<00:00, 34.23it/s]

Run #1 Results



```
=====
[1] ANALYZING RUN #2 / 20
```

```
Run ID: c5d5c6e14f254c949f63bf21385c795e
Final Validation Loss: 0.92605
```

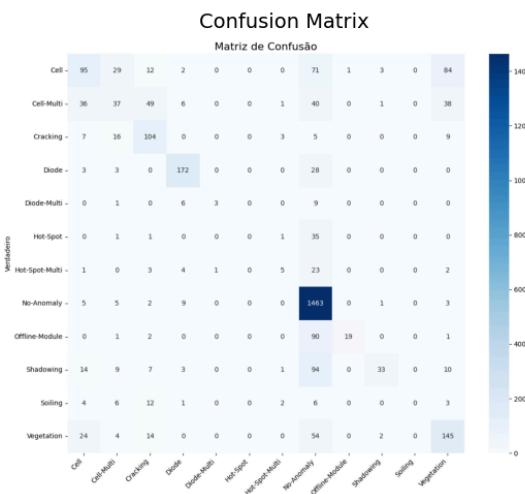
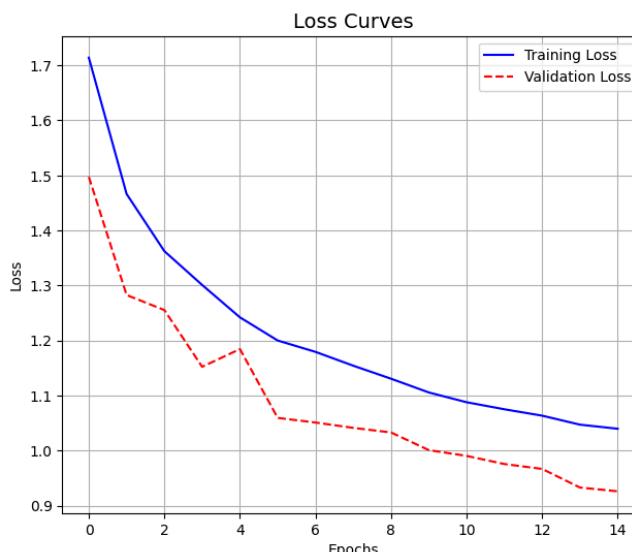
Parameters:

- lr: 0.0005659641838443741
- dropout_rate: 0.49911891667559455
- n_feature: 14

Downloading artifacts: 100%

1/1 [00:00<00:00, 43.91it/s]

Run #2 Results



```
=====
[1] ANALYZING RUN #3 / 20
```

```
Run ID: bdb6e2403e1f4a92becbd847a1d526d0
Final Validation Loss: 0.92858
```

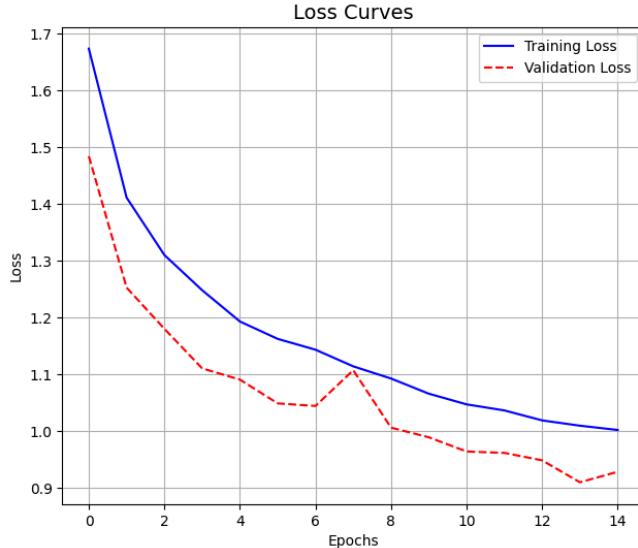
Parameters:

- lr: 0.0009560059228078452
- dropout_rate: 0.42824797097280914
- n_feature: 14

Downloading artifacts: 100%

1/1 [00:00<00:00, 50.02it/s]

Run #3 Results



Confusion Matrix

		Matriz de Confusão											
Verdadeiro	Previsto	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Soiling	Vegetation
		121	23	13	1	0	0	0	70	2	3	0	64
Cell	Cell	121	23	13	1	0	0	0	70	2	3	0	64
Cell-Multi	Cell-Multi	56	35	49	4	0	0	0	24	1	7	0	32
Cracking	Cracking	12	15	99	0	0	0	2	6	0	0	1	9
Diode	Diode	5	1	0	168	1	0	0	29	1	1	0	0
Diode-Multi	Diode-Multi	0	1	0	3	7	0	0	7	1	0	0	0
Hot-Spot	Hot-Spot	2	0	1	0	0	0	0	34	0	1	0	0
Hot-Spot-Multi	Hot-Spot-Multi	2	0	0	4	0	0	9	18	0	4	0	2
No-Anomaly	No-Anomaly	3	3	1	5	0	0	0	1	1470	2	2	0
Offline-Module	Offline-Module	0	2	2	1	0	0	0	1	84	23	0	0
Shadowing	Shadowing	16	10	4	2	0	0	0	101	0	28	0	10
Soiling	Soiling	5	3	13	1	0	0	3	5	0	0	0	4
Vegetation	Vegetation	32	11	11	1	0	0	0	64	0	3	0	111

ANALYZING RUN #4 / 20

Run ID: 0d17a1e4925e4be7a6b91bb60ffb0e66

Final Validation Loss: 0.96616

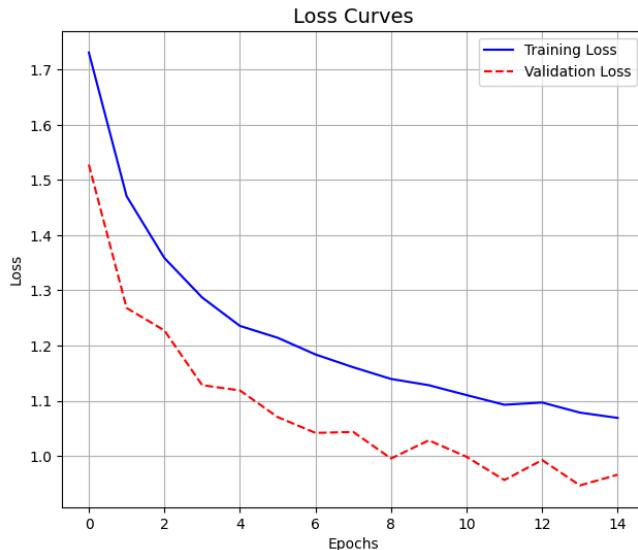
Parameters:

- lr: 0.0008893570067392511
- dropout_rate: 0.494771269228879
- n_feature: 15

Downloading artifacts: 100%

1/1 [00:00<00:00, 55.93it/s]

Run #4 Results



Confusion Matrix

		Matriz de Confusão											
Verdadeiro	Previsto	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Soiling	Vegetation
		121	18	8	0	0	0	0	80	2	3	0	65
Cell	Cell	121	18	8	0	0	0	0	80	2	3	0	65
Cell-Multi	Cell-Multi	59	37	35	3	0	0	0	39	0	2	0	37
Cracking	Cracking	14	15	94	0	0	0	1	9	0	0	0	11
Diode	Diode	2	1	0	157	0	0	0	45	0	1	0	0
Diode-Multi	Diode-Multi	0	0	0	7	2	0	0	10	0	0	0	0
Hot-Spot	Hot-Spot	1	1	0	0	0	0	0	36	0	0	0	0
Hot-Spot-Multi	Hot-Spot-Multi	1	2	1	1	0	0	0	11	21	0	0	2
No-Anomaly	No-Anomaly	3	3	1	5	0	0	0	0	1473	0	2	0
Offline-Module	Offline-Module	0	2	2	0	1	0	0	96	12	0	0	0
Shadowing	Shadowing	19	10	8	0	0	0	0	1	92	0	32	0
Soiling	Soiling	7	5	11	1	0	0	2	3	0	1	0	4
Vegetation	Vegetation	40	9	8	0	0	0	0	62	0	2	0	122

ANALYZING RUN #5 / 20

Run ID: 76a8d8c24fd47e39612c149bb7565f0

Final Validation Loss: 0.96672

Parameters:

- lr: 0.0002981888093655215
- dropout_rate: 0.4192161848024607
- n_feature: 13

Downloading artifacts: 100%

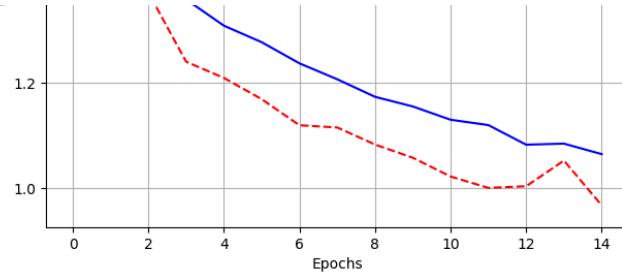
1/1 [00:00<00:00, 55.53it/s]

Run #5 Results



Confusion Matrix

		Matriz de Confusão											
Verdadeiro	Previsto	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Soiling	Vegetation
		99	24	15	2	0	0	0	79	1	4	0	73
Cell	Cell	99	24	15	2	0	0	0	79	1	4	0	73
Cell-Multi	Cell-Multi	40	37	58	6	0	0	0	34	0	1	0	32
Cracking	Cracking	9	10	106	1	0	0	0	1	10	0	0	7
Diode	Diode	3	2	0	172	0	0	0	28	0	1	0	0
Diode-Multi	Diode-Multi	0	1	0	5	7	0	0	5	1	0	0	0
Hot-Spot	Hot-Spot	1	1	2	0	0	0	0	32	0	0	0	2
Hot-Spot-Multi	Hot-Spot-Multi	2	0	2	5	0	0	0	7	19	0	1	3



No-Anomaly	4	6	6	7	0	0	0	0	1456	2	5	0	2
Offline-Module	1	1	3	0	0	0	0	0	83	24	0	0	1
Shadowing	11	5	13	2	0	0	0	1	97	0	31	0	11
Solling	5	5	12	1	0	0	0	3	4	0	1	0	3
Vegetation	27	4	15	0	0	0	0	0	54	0	2	0	141
Cell													
Cell-Multi													
Cracking													
Diode													
Diode-Multi													
Hot-Spot													
Hot-Spot-Multi													
No-Anomaly													
Offline-Module													
Shadowing													
Solling													
Vegetation													
Previsto													

=====

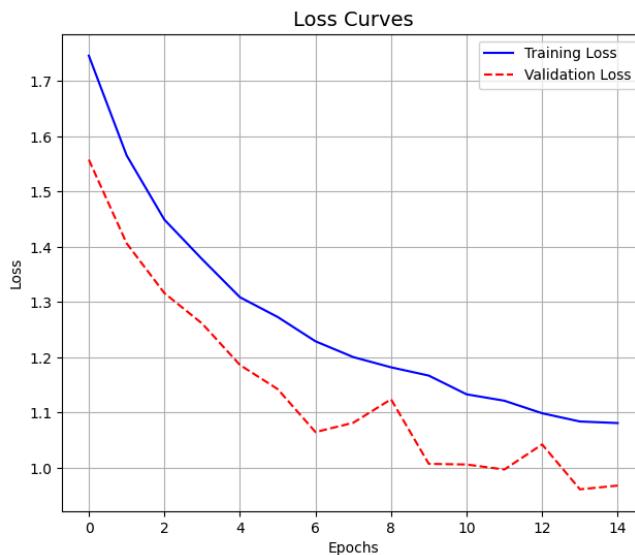
Analyzing Run #6 / 20
Run ID: 1280f0a12fc644058545eac74463e2a2
Final Validation Loss: 0.96787

Parameters:
- lr: 0.0009830175263413061
- dropout_rate: 0.4570030231706655
- n_feature: 16

Downloading artifacts: 100%

1/1 [00:00<00:00, 22.59it/s]

Run #6 Results



Confusion Matrix

Matriz de Confusão													
Verdadeiro	Previsto												
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation	
Cell	116	23	16	0	0	0	0	59	1	0	0	82	
Cell-Multi	61	27	55	1	0	0	0	28	0	2	0	34	
Cracking	11	14	102	0	0	0	0	7	0	0	0	30	
Diode	3	0	0	166	0	0	0	35	1	0	0	1	
Diode-Multi	0	1	0	7	1	0	0	7	3	0	0	0	
Hot-Spot	2	3	0	0	0	1	0	31	0	1	0	0	
Hot-Spot-Multi	0	3	0	5	0	0	0	22	0	0	0	4	
No-Anomaly	10	5	0	5	0	0	0	0	1465	0	3	0	
Offline-Module	1	0	2	0	0	0	0	91	18	0	0	1	
Shadowing	25	2	13	1	0	0	0	107	0	15	0	7	
Solling	6	2	16	1	0	0	0	5	0	0	0	3	
Vegetation	38	5	16	0	0	0	0	46	0	3	0	135	
Previsto													

=====

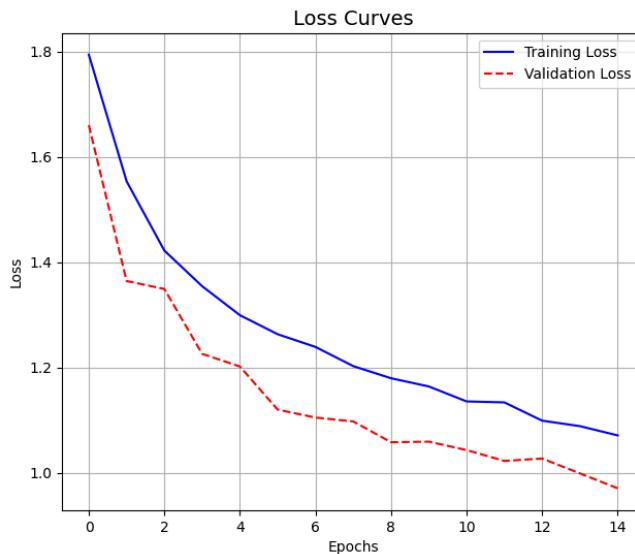
Analyzing Run #7 / 20
Run ID: 106fc20db31141abaad809efed48597e
Final Validation Loss: 0.97031

Parameters:
- lr: 0.0005178437215550134
- dropout_rate: 0.44139206692138394
- n_feature: 13

Downloading artifacts: 100%

1/1 [00:00<00:00, 50.90it/s]

Run #7 Results



Confusion Matrix

Matriz de Confusão													
Verdadeiro	Previsto												
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation	
Cell	108	24	13	2	0	0	0	82	2	3	0	63	
Cell-Multi	34	28	58	2	0	0	0	1	44	0	2	0	39
Cracking	5	11	105	0	0	0	0	1	32	0	2	0	8
Diode	1	3	0	177	0	0	0	25	0	0	0	0	0
Diode-Multi	0	1	1	4	4	0	0	8	1	0	0	0	0
Hot-Spot	1	0	2	0	0	0	0	34	0	0	0	1	
Hot-Spot-Multi	1	1	4	5	1	0	0	7	18	0	0	0	2
No-Anomaly	3	4	7	8	0	0	0	0	1461	2	2	0	1
Offline-Module	0	2	3	0	0	0	0	0	84	24	0	0	0
Shadowing	13	9	11	2	0	0	0	1	97	0	29	0	9
Solling	4	5	13	1	0	0	0	3	5	0	1	0	2
Vegetation	26	5	14	0	0	0	0	64	0	3	0	0	111
Previsto													

=====

Analyzing Run #8 / 20

<https://colab.research.google.com/drive/1a4eGTCmWszT7rurzb7XQ7nrt4o7tivv?authuser=1#scrollTo=VKDsCSvtIUcv&printMode=true>

Run ID: 881fd02b6d0244ffbb32076eef10795
Final Validation Loss: 0.98987

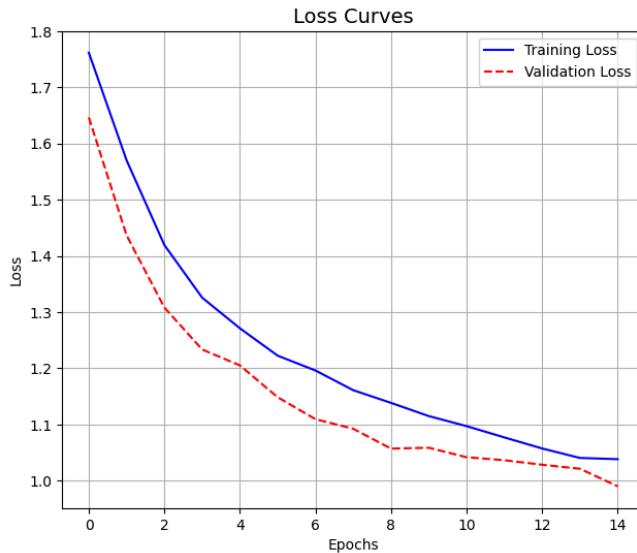
Parameters:

- lr: 0.0004435327396568911
- dropout_rate: 0.28365028413191185
- n_feature: 8

Downloading artifacts: 100%

1/1 [00:00<00:00, 51.54it/s]

Run #8 Results



Confusion Matrix

		Matriz de Confusão											
Avaliadores	Previsto	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation
		117	23	15	1	0	0	0	88	3	1	0	49
Cell	Cell	117	23	15	1	0	0	0	88	3	1	0	49
Cell-Multi	Cell-Multi	47	51	44	4	0	0	1	37	0	0	0	24
Cracking	Cracking	11	12	102	0	0	0	1	11	0	2	0	5
Diode	Diode	2	4	0	172	0	0	0	26	0	1	0	1
Diode-Multi	Diode-Multi	0	2	1	5	5	0	0	6	0	0	0	0
Hot-Spot	Hot-Spot	4	1	3	0	0	0	0	30	0	0	0	0
Hot-Spot-Multi	Hot-Spot-Multi	2	1	2	6	1	3	4	18	0	0	0	2
No-Anomaly	No-Anomaly	17	2	1	9	0	1	0	1451	2	2	0	3
Offline-Module	Offline-Module	1	0	3	0	0	0	0	84	25	0	0	0
Shadowing	Shadowing	13	3	12	2	0	0	0	90	0	37	0	12
Solling	Solling	6	6	10	2	0	0	1	5	0	2	1	1
Vegetation	Vegetation	42	8	14	1	0	0	0	64	0	3	0	111

=====

Analyzing Run #9 / 20

Run ID: 95733da7c9b842549f1d1902e911400d

Final Validation Loss: 1.00397

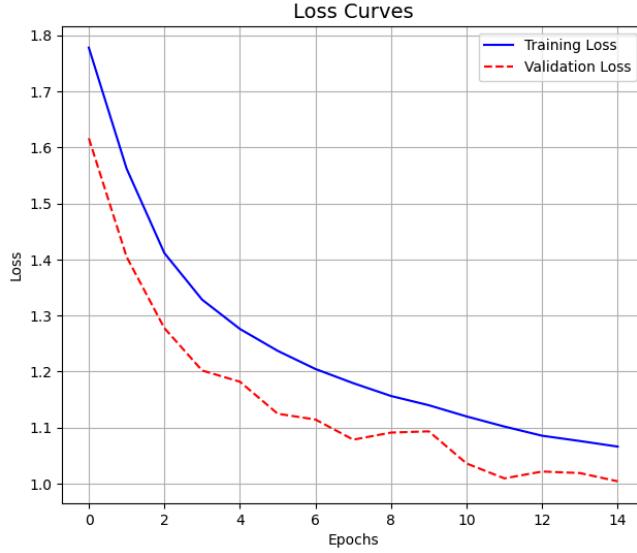
Parameters:

- lr: 0.0004584419071211875
- dropout_rate: 0.37420053111721596
- n_feature: 7

Downloading artifacts: 100%

1/1 [00:00<00:00, 41.56it/s]

Run #9 Results



Confusion Matrix

		Matriz de Confusão											
Avaliadores	Previsto	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation
		115	22	8	3	0	0	1	78	1	5	0	64
Cell	Cell	115	22	8	3	0	0	1	78	1	5	0	64
Cell-Multi	Cell-Multi	42	39	43	4	0	0	2	37	0	3	0	38
Cracking	Cracking	10	12	100	1	0	0	1	12	0	1	0	7
Diode	Diode	5	2	2	166	0	0	0	31	0	0	0	0
Diode-Multi	Diode-Multi	0	3	0	4	4	0	0	8	0	0	0	0
Hot-Spot	Hot-Spot	2	5	3	1	0	0	0	1	26	0	0	0
Hot-Spot-Multi	Hot-Spot-Multi	1	2	3	4	0	1	11	16	0	0	0	1
No-Anomaly	No-Anomaly	8	3	5	12	0	0	0	1	1457	0	2	0
Offline-Module	Offline-Module	1	1	2	0	0	0	0	0	101	8	0	0
Shadowing	Shadowing	15	2	12	1	0	0	0	3	110	0	17	1
Solling	Solling	5	5	12	1	0	0	0	1	5	0	1	2
Vegetation	Vegetation	39	10	16	0	0	0	2	66	0	2	0	108

=====

Analyzing Run #10 / 20

Run ID: d5446148350a4ffa9abfd8d4a216c251

Final Validation Loss: 1.00902

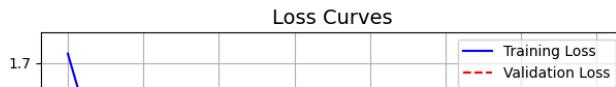
Parameters:

- lr: 0.0009776281491578342
- dropout_rate: 0.4735942343384408
- n_feature: 16

Downloading artifacts: 100%

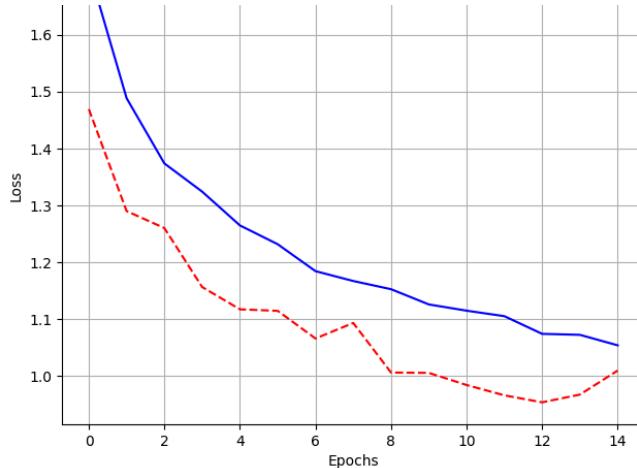
1/1 [00:00<00:00, 62.25it/s]

Run #10 Results



Confusion Matrix

		Matriz de Confusão											
Avaliadores	Previsto	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Solling	Vegetation
		109	37	18	1	0	0	0	68	0	1	0	63
Cell	Cell	109	37	18	1	0	0	0	68	0	1	0	63
Cell-Multi	Cell-Multi	42	39	43	4	0	0	2	37	0	3	0	38
Cracking	Cracking	10	12	100	1	0	0	1	12	0	1	0	7
Diode	Diode	5	2	2	166	0	0	0	31	0	0	0	0
Diode-Multi	Diode-Multi	0	3	0	4	4	0	0	8	0	0	0	0
Hot-Spot	Hot-Spot	2	5	3	1	0	0	0	1	26	0	0	0
Hot-Spot-Multi	Hot-Spot-Multi	1	2	3	4	0	1	11	16	0	0	0	1
No-Anomaly	No-Anomaly	8	3	5	12	0	0	0	1	1457	0	2	0
Offline-Module	Offline-Module	1	1	2	0	0	0	0	0	101	8	0	0
Shadowing	Shadowing	15	2	12	1	0	0	0	3	110	0	17	1
Solling	Solling	5	5	12	1	0	0	0	1	5	0	1	2
Vegetation	Vegetation	39	10	16	0	0	0	2	66	0	2	0	108



Cell-Multi	52	35	58	2	0	0	0	1	30	0	0	1	29
Cracking	12	10	104	0	0	0	0	11	0	0	0	0	7
Diode	3	1	2	153	0	0	0	47	0	0	0	0	0
Diode-Multi	0	1	0	6	2	0	0	10	0	0	0	0	0
Hot-Spot	2	4	1	0	0	0	0	31	0	0	0	0	0
Hot-Spot Multi	2	1	6	1	0	0	0	6	21	0	0	0	2
No-Anomaly	7	7	4	9	0	0	0	3	1454	1	1	0	2
Offline-Module	3	1	3	1	0	0	0	1	91	13	0	0	0
Shadowing	14	12	14	3	0	0	0	3	105	0	12	0	8
Solling	5	2	14	1	0	0	0	2	6	0	1	1	2
Vegetation	37	10	22	0	0	0	0	1	53	1	1	0	118
Previsto	0	0	0	0	0	0	0	0	0	0	0	0	0

```
=====
 ANALYZING RUN #11 / 20
Run ID: 4c32990dd1b1427fb361593eae4f912c
Final Validation Loss: 1.02700
```

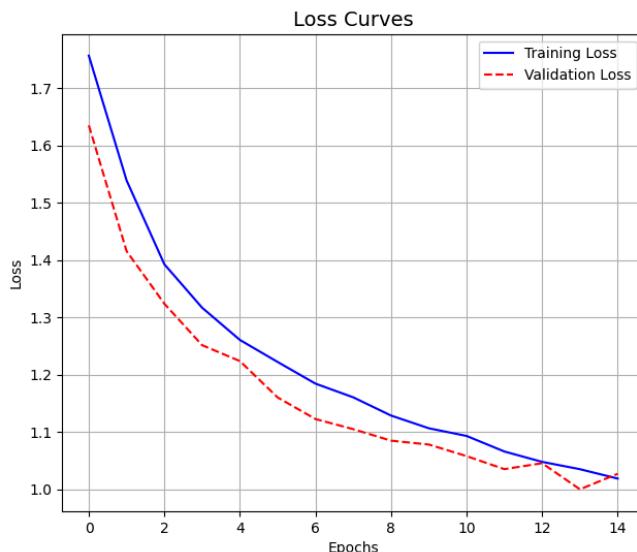
Parameters:

- lr: 0.0002432400716842307
- dropout_rate: 0.23448094067692607
- n_feature: 9

Downloading artifacts: 100%

1/1 [00:00<00:00, 50.36it/s]

Run #11 Results



Confusion Matrix

Matriz de Confusão													
Cell	90	37	8	4	0	0	0	79	3	5	0	0	71
Cell-Multi	27	46	56	4	0	0	0	33	1	4	0	0	37
Cracking	4	15	100	2	0	0	0	2	8	0	0	1	13
Diode	4	2	0	173	0	0	0	24	0	2	0	0	1
Diode-Multi	0	4	2	4	5	0	0	3	0	0	0	0	1
Hot-Spot	3	4	3	1	0	3	0	23	0	0	0	0	1
Hot-Spot Multi	1	1	4	2	1	4	11	12	0	0	0	0	3
No-Anomaly	13	10	6	9	1	0	0	3	1430	8	5	0	3
Offline-Module	1	4	2	0	1	0	0	3	63	35	1	0	3
Shadowing	13	6	10	2	0	0	0	3	89	0	37	0	11
Solling	3	3	9	1	0	0	0	2	7	0	2	5	2
Vegetation	17	6	16	2	0	0	0	1	48	2	3	1	347
Previsto	0	0	0	0	0	0	0	0	0	0	0	0	0

```
=====
 ANALYZING RUN #12 / 20
Run ID: 463cf883dbb64a41a1b2a3f683d031b6
Final Validation Loss: 1.03221
```

Parameters:

- lr: 0.0008343190339022007
- dropout_rate: 0.4940183698244794
- n_feature: 16

Downloading artifacts: 100%

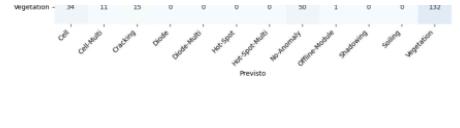
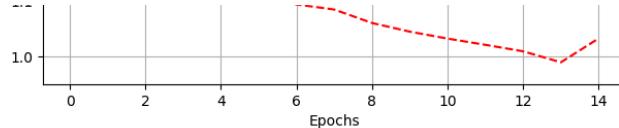
1/1 [00:00<00:00, 37.97it/s]

Run #12 Results



Confusion Matrix

Matriz de Confusão													
Cell	94	37	18	0	0	0	0	67	0	0	0	0	81
Cell-Multi	41	47	50	3	0	0	0	36	0	1	0	0	30
Cracking	8	17	96	2	0	0	0	8	0	0	0	0	13
Diode	0	1	0	169	0	0	0	34	0	1	0	0	1
Diode-Multi	0	2	1	8	1	0	0	6	1	0	0	0	0
Hot-Spot	2	2	1	0	0	0	0	33	0	0	0	0	0
Hot-Spot Multi	1	4	3	7	0	0	0	21	0	0	0	0	2
No-Anomaly	9	10	0	10	0	0	0	0	1456	0	2	0	1
Offline-Module	0	4	3	1	0	0	0	0	96	9	0	0	0
Shadowing	15	15	12	2	0	0	0	0	104	0	19	0	4
Solling	2	6	15	1	0	0	0	7	0	0	0	0	3
Vegetation	0	0	0	0	0	0	0	0	0	0	0	0	0
Previsto	0	0	0	0	0	0	0	0	0	0	0	0	0



=====

ANALYZING RUN #13 / 20
Run ID: 0c9a0510fe82430ba1c128848f665823
Final Validation Loss: 1.16689

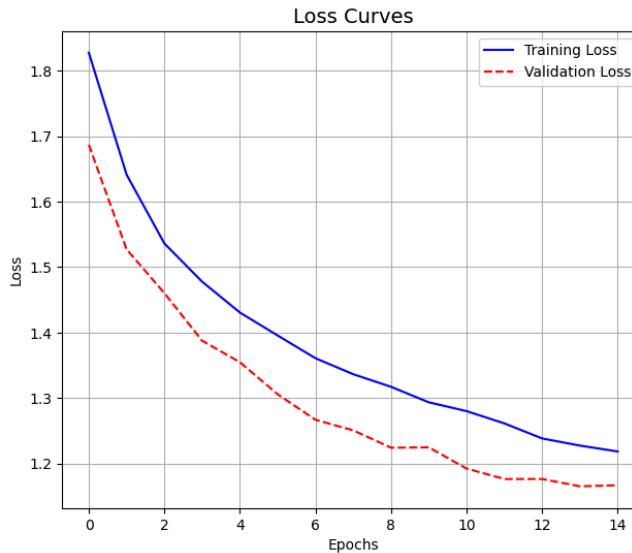
Parameters:

- lr: 0.00013600333114777974
- dropout_rate: 0.4147736431683601
- n_feature: 12

Downloading artifacts: 100%

1/1 [00:00<00:00, 54.69it/s]

Run #13 Results



Confusion Matrix

Matriz de Confusão

	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Soiling	Vegetation	Previsto
Cell	62	21	17	8	0	0	0	111	0	6	0	72	
Cell-Multi	30	20	66	11	0	0	0	47	0	3	0	31	
Cracking	6	11	102	5	0	0	0	10	0	1	0	9	
Diode	1	4	2	168	0	0	0	31	0	0	0	0	
Diode-Multi	0	1	3	9	0	0	0	6	0	0	0	0	
Hot-Spot	0	2	6	0	0	0	0	1	27	0	0	0	
Hot-Spot-Multi	1	0	7	8	0	0	0	3	17	0	0	3	
No-Anomaly	14	16	15	12	0	0	0	1	1416	0	7	0	
Offline-Module	0	6	5	1	0	0	0	97	0	0	0	4	
Shadowing	8	4	15	3	0	0	0	102	0	27	0	12	
Soiling	1	4	11	1	0	0	0	7	0	6	0	4	
Vegetation	22	6	16	1	0	0	0	70	0	1	0	127	

=====

ANALYZING RUN #14 / 20

Run ID: 1e7e554b2af24527970edcefbd0f0fa2
Final Validation Loss: 1.17887

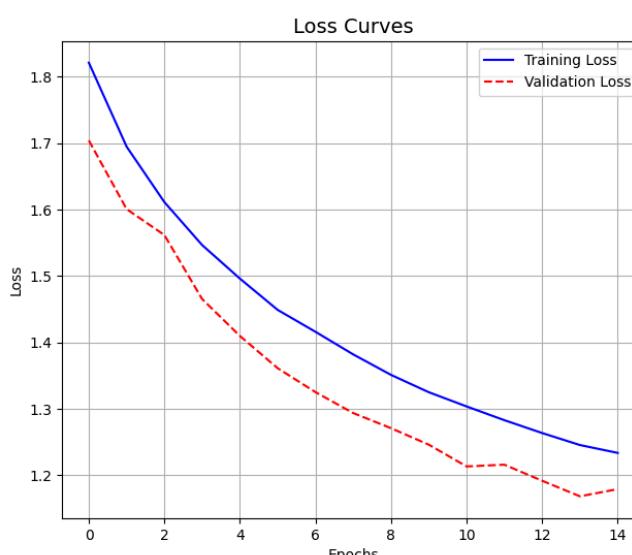
Parameters:

- lr: 0.0002480318287915911
- dropout_rate: 0.456812612682163
- n_feature: 6

Downloading artifacts: 100%

1/1 [00:00<00:00, 39.59it/s]

Run #14 Results



Confusion Matrix

Matriz de Confusão

	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Soiling	Vegetation	Previsto
Cell	55	25	23	5	0	0	0	117	0	0	0	72	
Cell-Multi	21	19	55	7	0	0	1	60	0	1	0	44	
Cracking	3	4	108	4	0	0	0	14	0	0	0	11	
Diode	1	2	5	156	0	0	0	40	0	0	0	2	
Diode-Multi	0	0	3	9	0	0	0	7	0	0	0	0	
Hot-Spot	0	0	5	0	0	0	0	31	0	0	0	2	
Hot-Spot-Multi	0	2	9	7	0	0	0	3	16	0	0	2	
No-Anomaly	6	6	26	7	0	0	0	0	1437	0	1	0	
Offline-Module	0	3	6	1	0	0	0	100	1	0	0	2	
Shadowing	6	3	16	2	0	0	1	122	0	8	0	13	
Soiling	1	4	19	1	0	0	0	6	0	1	0	2	
Vegetation	15	4	17	0	0	0	0	82	0	1	0	124	

=====

ANALYZING RUN #15 / 20

Run ID: 26adb64115a417b9abcd74eff6dded
Final Validation Loss: 1.22086

Parameters:

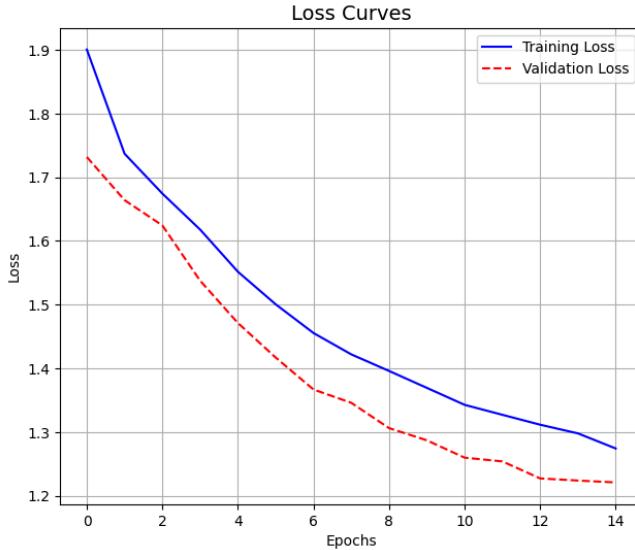
- lr: 7.089770147367083e-05
- dropout_rate: 0.3796657127177262

- n_feature: 11

Downloading artifacts: 100%

1/1 [00:00<00:00, 55.57it/s]

Run #15 Results



Confusion Matrix

		Matriz de Confusão												
		Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Siling	Vegetation	
Verdadeiro	Previsto	Cell	36	20	16	10	0	0	0	152	0	3	0	60
		Cell-Multi	20	14	69	13	0	0	0	48	2	2	0	40
Verdadeiro	Previsto	Cracking	1	6	107	6	0	0	0	35	0	3	0	6
		Diode	0	2	3	149	0	0	0	49	0	2	0	1
Verdadeiro	Previsto	Diode-Multi	0	0	4	9	0	0	0	6	0	0	0	0
		Hot-Spot	1	1	3	0	0	0	1	30	0	0	0	2
Verdadeiro	Previsto	Hot-Spot-Multi	0	1	9	4	0	0	3	28	0	2	0	2
		No-Anomaly	8	10	26	6	0	0	0	1424	0	3	0	8
Verdadeiro	Previsto	Offline-Module	0	1	8	1	0	0	0	93	4	0	0	6
		Shadowing	11	5	16	3	0	0	1	110	0	15	0	10
Verdadeiro	Previsto	Siling	1	5	11	4	0	0	0	9	0	2	0	2
		Vegetation	11	1	22	3	0	0	1	87	0	1	0	117

=====

Analyzing Run #16 / 20

Run ID: 1e66885640ae4841a45e4ff3b19ee027

Final Validation Loss: 1.27195

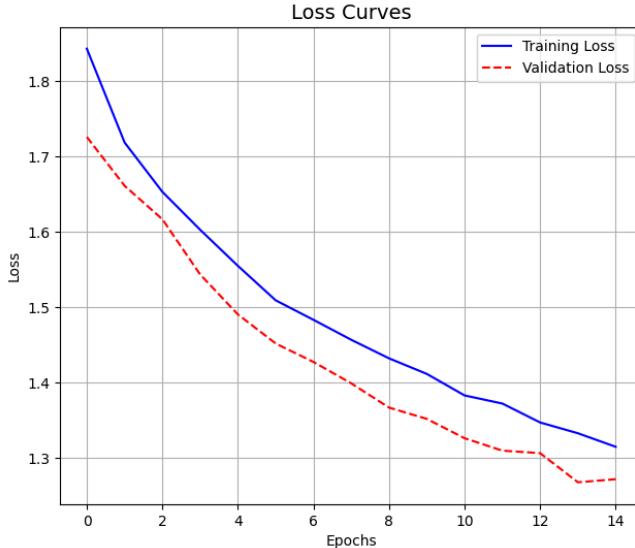
Parameters:

- lr: 0.00015865208232788787
- dropout_rate: 0.28238038153830614
- n_feature: 4

Downloading artifacts: 100%

1/1 [00:00<00:00, 57.86it/s]

Run #16 Results



Confusion Matrix

		Matriz de Confusão												
		Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Siling	Vegetation	
Verdadeiro	Previsto	Cell	48	25	12	5	0	0	0	154	0	0	0	53
		Cell-Multi	23	21	45	11	0	0	0	78	0	0	0	30
Verdadeiro	Previsto	Cracking	2	10	102	1	0	0	0	23	0	0	0	6
		Diode	2	3	5	132	0	0	0	63	0	0	0	1
Verdadeiro	Previsto	Diode-Multi	0	0	2	6	0	0	0	11	0	0	0	0
		Hot-Spot	0	0	4	0	0	0	0	33	0	0	0	1
Verdadeiro	Previsto	Hot-Spot-Multi	1	3	5	6	0	0	3	20	0	0	0	1
		No-Anomaly	7	7	19	3	0	0	0	1443	0	1	0	8
Verdadeiro	Previsto	Offline-Module	2	2	6	1	0	0	0	101	0	0	0	1
		Shadowing	5	4	13	3	0	0	0	125	0	7	0	13
Verdadeiro	Previsto	Siling	3	6	10	1	0	0	1	9	0	1	3	0
		Vegetation	20	18	14	1	0	0	0	104	0	0	0	86

=====

Analyzing Run #17 / 20

Run ID: a8de8d76d1e9435cb3fc557ff8bd817

Final Validation Loss: 1.32000

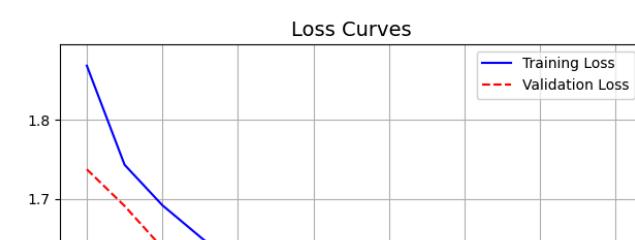
Parameters:

- lr: 0.0001052863640327261
- dropout_rate: 0.3424898657595308
- n_feature: 5

Downloading artifacts: 100%

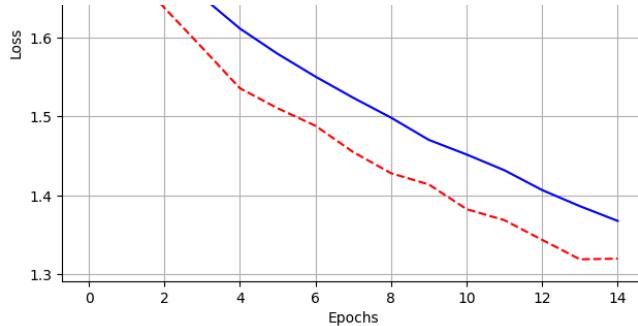
1/1 [00:00<00:00, 37.28it/s]

Run #17 Results



Confusion Matrix

		Matriz de Confusão												
		Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Siling	Vegetation	
Verdadeiro	Previsto	Cell	34	9	15	9	0	0	0	175	0	1	0	54
		Cell-Multi	24	8	49	14	0	0	1	77	0	0	0	35
Verdadeiro	Previsto	Cracking	5	4	99	4	0	0	1	24	0	1	0	6
		Diode	2	1	4	133	0	0	0	65	0	0	0	1
Verdadeiro	Previsto	Diode-Multi	0	0	3	6	0	0	0	10	0	0	0	0
		Hot-Spot	0	0	0	0	0	0	0	44	0	0	0	0
Verdadeiro	Previsto	Hot-Spot-Multi	0	0	0	0	0	0	0	10	0	0	0	0
		No-Anomaly	0	0	0	0	0	0	0	1443	0	1	0	8
Verdadeiro	Previsto	Offline-Module	0	0	0	0	0	0	0	101	0	0	0	1
		Shadowing	0	0	0	0	0	0	0	125	0	7	0	13
Verdadeiro	Previsto	Siling	0	0	0	0	0	0	0	9	0	1	3	0
		Vegetation	0	0	0	0	0	0	0	104	0	0	0	86



Variables	Previsto											
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Selling	Vegetation
Hot-Spot-Multi	0	0	7	9	0	0	0	21	0	0	0	0
No-Anomaly	15	7	20	6	0	0	0	1430	0	1	0	9
Offline-Module	5	0	4	3	0	0	0	98	0	0	0	3
Shadowing	7	1	16	1	0	0	0	128	0	5	0	13
Selling	1	1	13	3	0	0	0	15	0	0	0	1
Vegetation	15	2	15	4	0	0	1	101	0	0	0	105

=====

ANALYZING RUN #18 / 20
Run ID: 7df911812b949989df2a6d2e41d08f5
Final Validation Loss: 1.38757

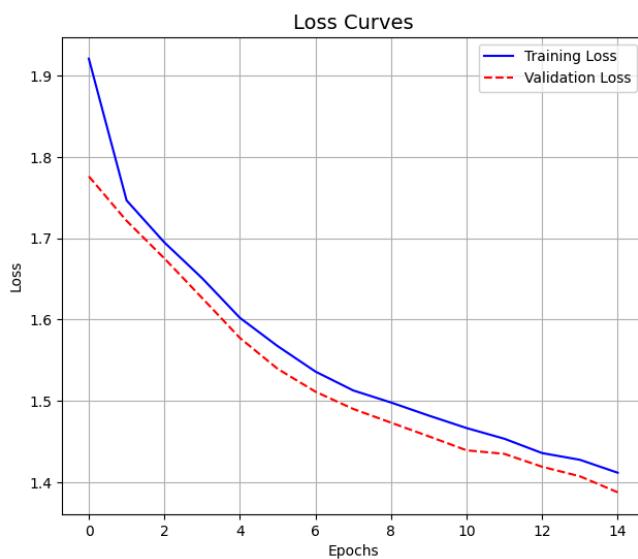
Parameters:

- lr: 4.102070668045529e-05
- dropout_rate: 0.10187959573942883
- n_feature: 5

Downloading artifacts: 100%

1/1 [00:00<00:00, 50.98it/s]

Run #18 Results



Variables	Previsto											
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Selling	Vegetation
Cell	22	6	2	7	0	0	0	223	0	3	0	34
Cell-Multi	12	7	26	15	0	0	1	115	1	2	0	29
Cracking	5	7	66	7	0	1	2	48	0	0	0	8
Diode	1	1	2	123	0	0	0	76	0	0	0	3
Diode-Multi	0	0	2	6	0	0	0	10	0	0	0	1
Hot-Spot	0	0	3	0	0	0	0	34	0	0	0	1
Hot-Spot-Multi	0	0	3	8	0	0	6	22	0	0	0	0
No-Anomaly	4	4	5	5	0	0	2	1461	0	1	0	6
Offline-Module	0	0	5	3	0	0	0	102	3	0	0	0
Shadowing	2	2	8	2	0	0	1	133	0	10	0	13
Selling	1	2	8	3	0	0	1	18	0	0	0	1
Vegetation	10	3	6	4	0	0	1	146	0	0	0	73

=====

ANALYZING RUN #19 / 20
Run ID: c6b51141424f469e9b9f01f792e999f8
Final Validation Loss: 1.56895

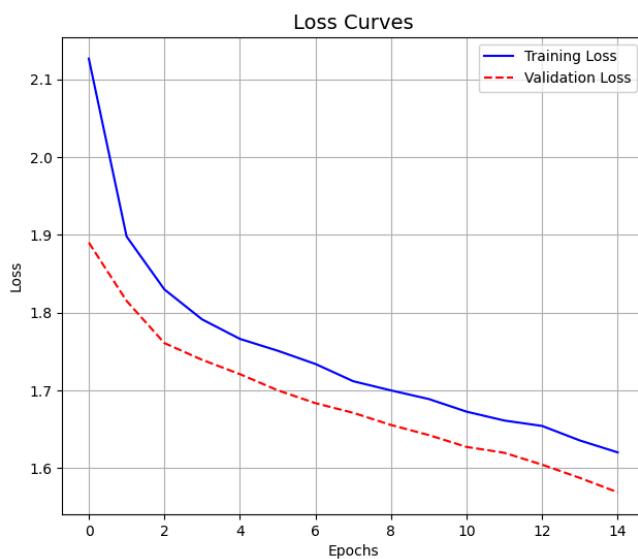
Parameters:

- lr: 2.098088282695299e-05
- dropout_rate: 0.38134458534086124
- n_feature: 7

Downloading artifacts: 100%

1/1 [00:00<00:00, 17.58it/s]

Run #19 Results



Variables	Previsto											
	Cell	Cell-Multi	Cracking	Diode	Diode-Multi	Hot-Spot	Hot-Spot-Multi	No-Anomaly	Offline-Module	Shadowing	Selling	Vegetation
Cell	10	0	0	1	0	0	0	252	0	0	0	34
Cell-Multi	10	0	0	3	0	0	0	173	0	0	0	22
Cracking	3	1	0	3	0	0	0	117	0	0	0	20
Diode	0	0	0	24	0	0	0	180	0	0	0	2
Diode-Multi	0	0	0	5	0	0	0	14	0	0	0	0
Hot-Spot	0	0	0	0	0	0	0	38	0	0	0	0
Hot-Spot-Multi	0	0	0	4	0	0	0	35	0	0	0	0
No-Anomaly	0	0	0	0	0	0	0	1477	0	0	0	11
Offline-Module	0	0	0	0	0	0	0	113	0	0	0	0
Shadowing	0	0	1	0	0	0	0	158	0	2	0	30
Selling	2	0	0	1	0	0	0	30	0	0	0	1
Vegetation	7	0	0	1	0	0	0	160	0	0	0	75

```
=====
1 ANALYZING RUN #20 / 20
Run ID: 13a251cf94e24bab936c20a4f2d612a9
Final Validation Loss: 1.68672
```

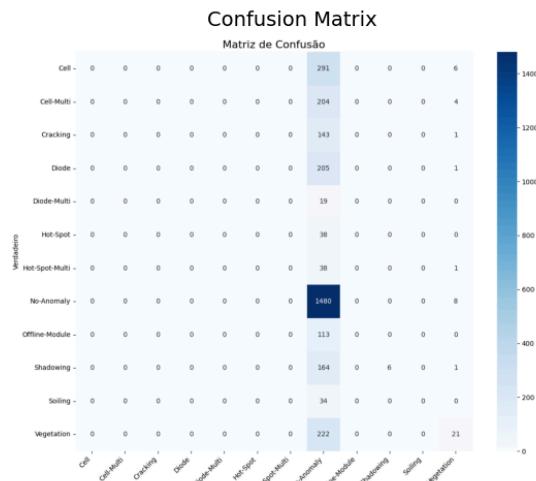
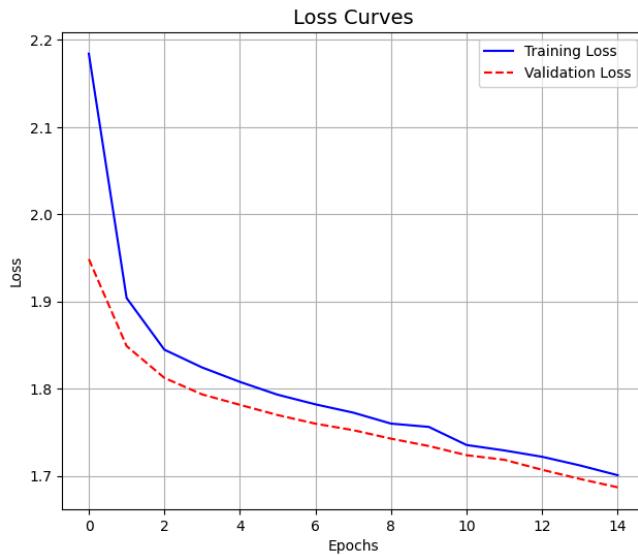
Parameters:

- lr: 1.1177285796090666e-05
- dropout_rate: 0.16989998387525254
- n_feature: 7

Downloading artifacts: 100%

1/1 [00:00<00:00, 38.19it/s]

Run #20 Results



```
=====
✓ Full report generated successfully!
```


✓ Rodando o melhor modelo

```

1 torch.manual_seed(42)
2
3 # Model/Architecture
4 model_cnn2_best_parameters = CNN2(n_feature=14, p=0.3371287734951104)
5
6 # Loss function
7 multi_loss_fn_best_parameters = nn.CrossEntropyLoss(reduction='mean')
8
9 # Optimizer
10 optimizer_cnn2_best_parameters = optim.Adam(model_cnn2_best_parameters.parameters(), lr=0.0004788645100667)

```

```
1 optimizer_cnn2_best_parameters.state_dict()
```

```
⤵ {'state': {},  
 'param_groups': [{}{'lr': 0.0004788645100667,  
 'betas': (0.9, 0.999),  
 'eps': 1e-08,  
 'weight_decay': 0,  
 'amsgrad': False,  
 'maximize': False,  
 'foreach': None,  
 'capturable': False,  
 'differentiable': False,  
 'fused': None,  
 'params': [0, 1, 2, 3, 4, 5, 6, 7]}]}
```

```

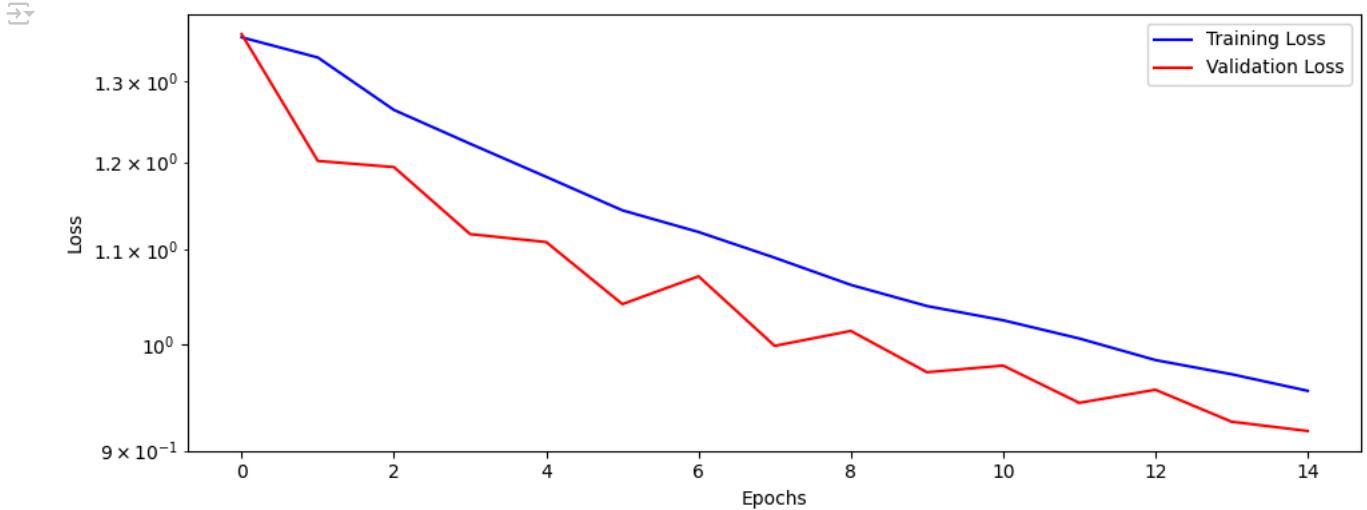
1 arch_cnn2_best_parameters = Architecture(model_cnn2_best_parameters,
2                               multi_loss_fn_best_parameters,
3                               optimizer_cnn2_best_parameters)
4 arch_cnn2_best_parameters.set_loaders(train_loader, val_loader)
5 arch_cnn2_best_parameters.train(15)

```

```
1 arch_cnn2_best_parameters.count_parameters()
```

```
⤵ 72832
```

```
1 fig = arch_cnn2_best_parameters.plot_losses()
```



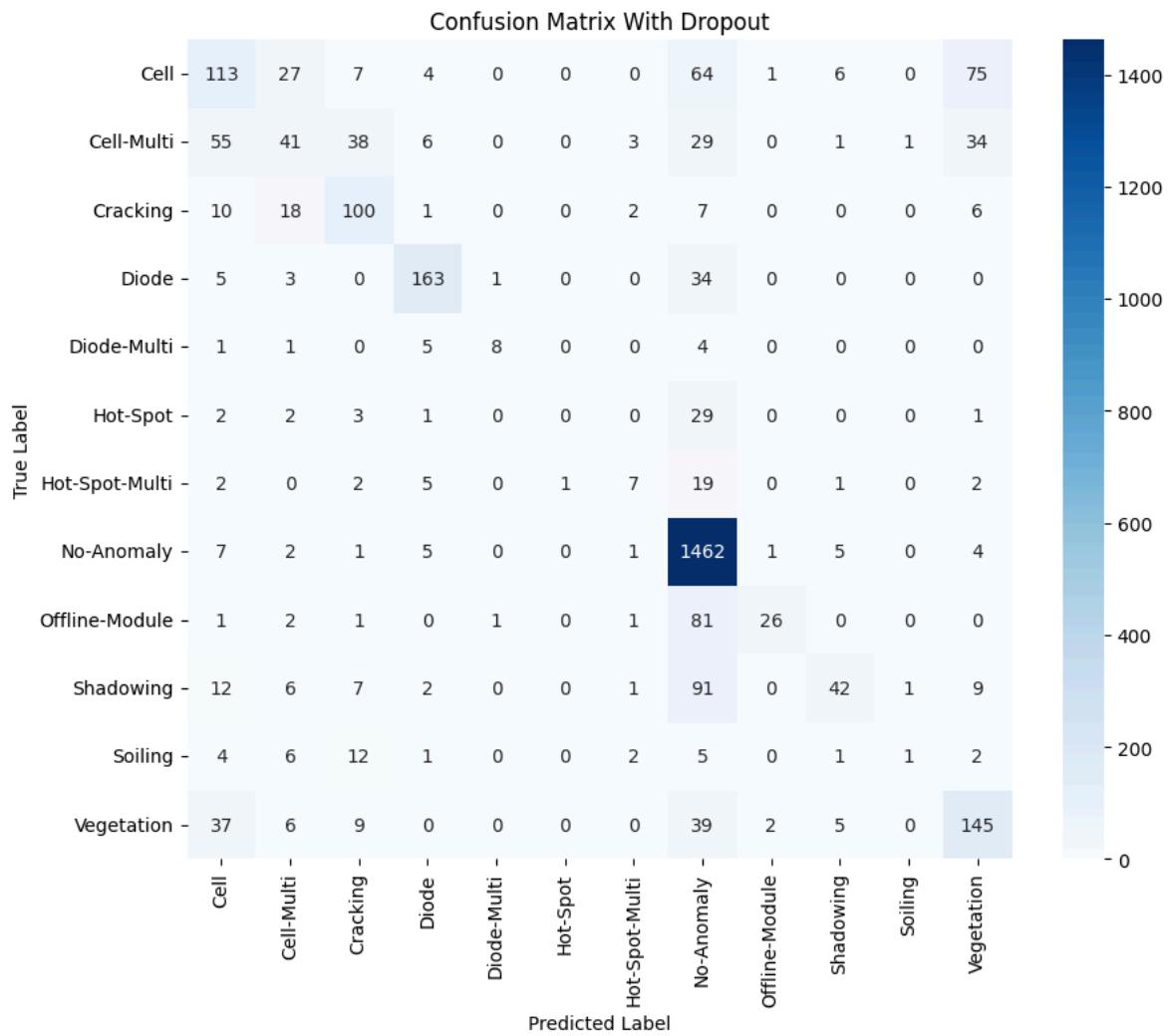
```

1 acc_train_cnn2bp = (lambda x: x[0].item() / x[1].item())(Architecture.loader_apply(train_loader, arch_cnn2_best_parameters.correct))
2 acc_val_cnn2bp = (lambda x: x[0].item() / x[1].item())(Architecture.loader_apply(val_loader, arch_cnn2_best_parameters.correct)).sum()
3
4 print(f"Acurácia de treinamento do modelo:{acc_train_cnn2bp: .5f}")
5 print(f"Acurácia de validação do modelo:{acc_val_cnn2bp: .5f}")

```

```
⤵ Acurácia de treinamento do modelo: 0.74441  
Acurácia de validação do modelo: 0.70267
```

```
1 cm_arch_cnn2bp = plot_confusion_matrix(arch_cnn2_best_parameters)
```



```

1 precision_per_class_cnn2bp = class_precision(cm_arch_cnn2bp)
2 print("\nPrecisão por Classe (derivada da matriz de confusão):")
3 class_names = val_data.classes # Assuming val_data.classes holds the class names
4 for i, precision in enumerate(precision_per_class_cnn2bp):
5     print(f" Classe '{class_names[i]}': {precision:.4f}")

```

```

[{"cell": "Cell", "precision": 0.4538}, {"cell": "Cell-Multi", "precision": 0.3596}, {"cell": "Cracking", "precision": 0.5556}, {"cell": "Diode", "precision": 0.8446}, {"cell": "Diode-Multi", "precision": 0.800}, {"cell": "Hot-Spot", "precision": 0.000}, {"cell": "Hot-Spot-Multi", "precision": 0.4118}, {"cell": "No-Anomaly", "precision": 0.7843}, {"cell": "Offline-Module", "precision": 0.8667}, {"cell": "Shadowing", "precision": 0.6885}, {"cell": "Soiling", "precision": 0.3333}, {"cell": "Vegetation", "precision": 0.5216}

```

```

1 class_counts_val = np.bincount(val_data.targets)
2 weighted_avg_precision_cnn2bp = weighted_precision(cm_arch_cnn2bp, class_counts_val)
3 print(f"\nMédia ponderada da precisão (ponderada pela quantidade de amostras): {weighted_avg_precision_cnn2bp:.4f}")

```

Média ponderada da precisão (ponderada pela quantidade de amostras): 0.6719

Modelo CNN com mais camadas

Implementação da VeryDeepCNN

```

1 class VeryDeepCNN(nn.Module):
2     def __init__(self, n_feature, p=0.0):
3         super(VeryDeepCNN, self).__init__()

```

```

1 self.n_feature = n_feature
2 self.p = p
3
4 # Creates the convolution layers
5 # Using padding=1 to better preserve the image size,
6 # which is essential for deeper networks.
7 self.conv1 = nn.Conv2d(in_channels=3, out_channels=n_feature, kernel_size=3, padding=1)
8 self.conv2 = nn.Conv2d(in_channels=n_feature, out_channels=n_feature * 2, kernel_size=3, padding=1)
9 self.conv3 = nn.Conv2d(in_channels=n_feature * 2, out_channels=n_feature * 4, kernel_size=3, padding=1)
10 self.conv4 = nn.Conv2d(in_channels=n_feature * 4, out_channels=n_feature * 8, kernel_size=3, padding=1)
11
12 # Creates the linear layers
13 # The input size of the first fully connected layer is calculated based on the output size of the last convolutional layer.
14 # For a 46x46 input image:
15 # After conv1 (k=3, p=1): (46 - 3 + 2*1) + 1 = 46. After pool (k=2): floor(46/2) = 23 -> 23x23
16 # After conv2 (k=3, p=1): (23 - 3 + 2*1) + 1 = 23. After pool (k=2): floor(23/2) = 11 -> 11x11
17 # After conv3 (k=3, p=1): (11 - 3 + 2*1) + 1 = 11. After pool (k=2): floor(11/2) = 5 -> 5x5
18 # After conv4 (k=3, p=1): (5 - 3 + 2*1) + 1 = 5. After pool (k=2): floor(5/2) = 2 -> 2x2
19 # So the flattened output size is (n_feature * 8) * 2 * 2
20 self.fc1 = nn.Linear(in_features=(n_feature * 8) * 4, out_features=50)
21 self.fc2 = nn.Linear(in_features=50, out_features=12)
22
23 # Creates dropout layer
24 self.drop = nn.Dropout(self.p)
25
26 def featurizer(self, x):
27     # Featurizer
28     # First convolutional block
29     # 3@46x46 -> n_feature@23x23
30     x = F.max_pool2d(F.relu(self.conv1(x)), kernel_size=2, stride=2)
31     # Second convolutional block
32     # n_feature@23x23 -> (n_feature*2)@11x11
33     x = F.max_pool2d(F.relu(self.conv2(x)), kernel_size=2, stride=2)
34     # Third convolutional block
35     # (n_feature*2)@11x11 -> (n_feature*4)@5x5
36     x = F.max_pool2d(F.relu(self.conv3(x)), kernel_size=2, stride=2)
37     # Fourth convolutional block
38     # (n_feature*4)@5x5 -> (n_feature*8)@2x2
39     x = F.max_pool2d(F.relu(self.conv4(x)), kernel_size=2, stride=2)
40
41     # Flatten the tensor for the classifier
42     # Input dimension ((n_feature*8)@2x2)
43     # Output dimension ((n_feature*8) * 4)
44     x = nn.Flatten()(x)
45
46     return x
47
48 def classifier(self, x):
49     # Classifier
50     # Hidden Layer
51     # Input dimension ((n_feature * 8) * 4)
52     # Output dimension (50)
53     if self.p > 0:
54         x = self.drop(x)
55     x = F.relu(self.fc1(x))
56
57     # Output Layer
58     # Input dimension (50)
59     # Output dimension (12)
60     if self.p > 0:
61         x = self.drop(x)
62     x = self.fc2(x)
63
64     return x
65
66 def forward(self, x):
67     x = self.featurizer(x)
68     x = self.classifier(x)
69
70     return x
71
72
73 torch.manual_seed(42)
74
75 model_very_deep = VeryDeepCNN(n_feature=5, p=0.3)
76 multi_loss_fn = nn.CrossEntropyLoss(reduction='mean')
77 optimizer_very_deep = optim.Adam(model_very_deep.parameters(), lr=3e-4)
78
79 #Training
80
81 arch_very_deep = Architecture(model_very_deep,
82                               multi_loss_fn,
83                               optimizer_very_deep)
84
85 arch_very_deep.set_loaders(train_loader, val_loader)
86
```

```
15 print("---- Training the Deeper Network (VeryDeepCNN) ---")
16 arch_very_deep.train(10) #number of epochs
17 print("---- Training Complete ---")
```

→ --- Training the Deeper Network (VeryDeepCNN) ---
--- Training Complete ---

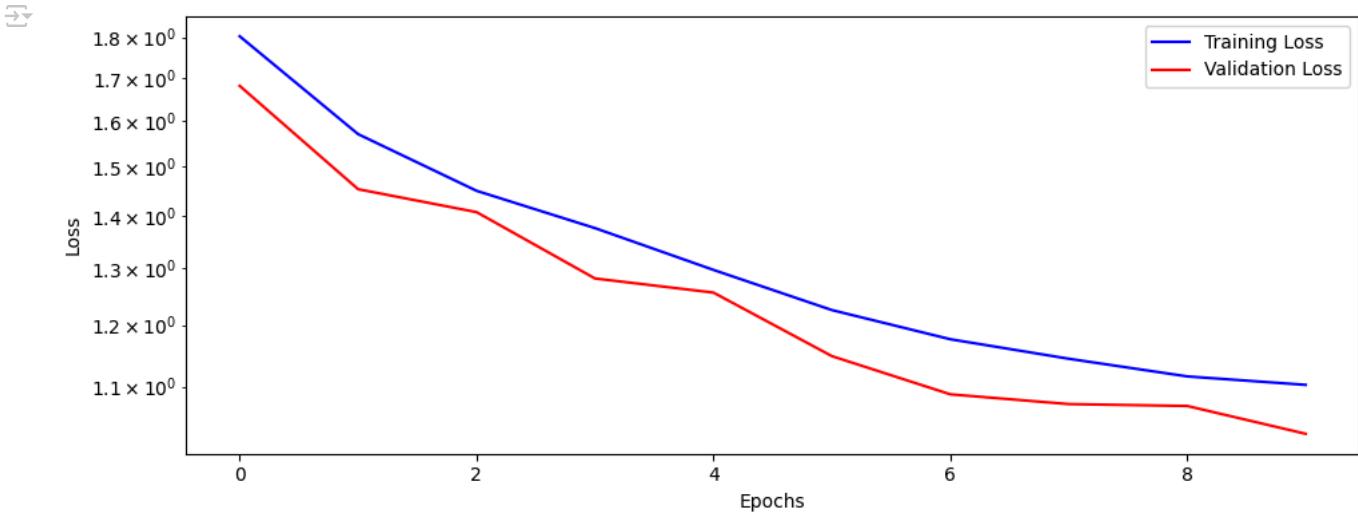
```
1 optimizer_very_deep.state_dict()
```

→ Mostrar saída oculta

```
1 arch_very_deep.count_parameters()
```

→ 18322

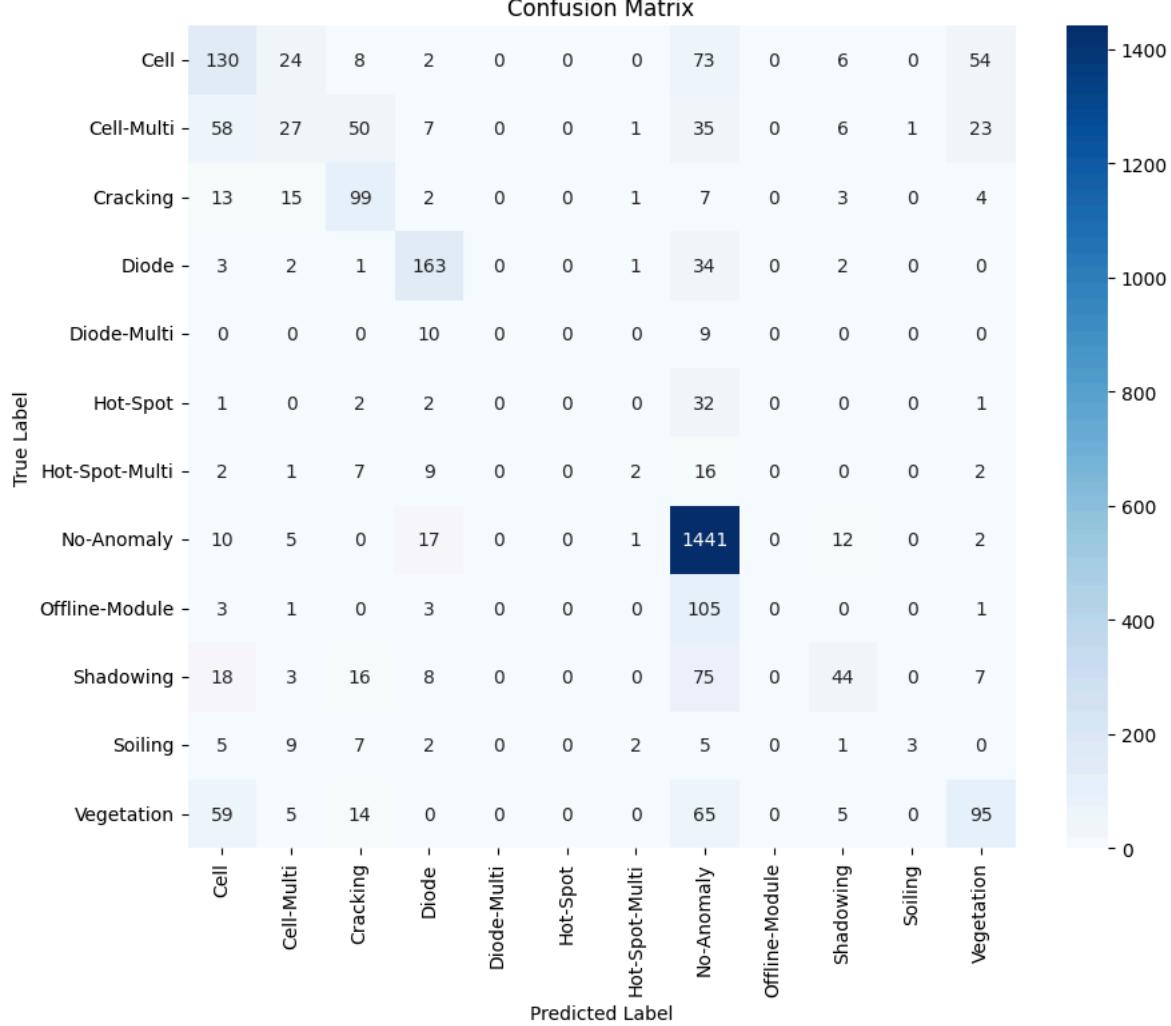
```
1 fig = arch_very_deep.plot_losses()
```



```
1 acc_train_vdd = (lambda x: x[0].item() / x[1].item())(Architecture.loader_apply(train_loader, arch_very_deep.correct).sum(axis=0))
2 acc_val_vdd = (lambda x: x[0].item() / x[1].item())(Architecture.loader_apply(val_loader, arch_very_deep.correct).sum(axis=0))
3
4 print(f"Acurácia de treinamento do modelo:{acc_train_vdd: .5f}")
5 print(f"Acurácia de validação do modelo:{acc_val_vdd: .3f}")
```

→ Acurácia de treinamento do modelo: 0.68976
Acurácia de validação do modelo: 0.668

```
1 cm_arch_vdd = plot_confusion_matrix(arch_very_deep)
```



```

1 precision_per_class_vdd = class_precision(cm_arch_vdd)
2 print("\nPrecisão por Classe (derivada da matriz de confusão):")
3 class_names = val_data.classes # Assuming val_data.classes holds the class names
4 for i, precision in enumerate(precision_per_class_vdd):
5     print(f" Classe '{class_names[i]}': {precision:.4f}")

```

→

```

Precisão por Classe (derivada da matriz de confusão):
Classe 'Cell': 0.4305
Classe 'Cell-Multi': 0.2935
Classe 'Cracking': 0.4853
Classe 'Diode': 0.7244
Classe 'Diode-Multi': 0.0000
Classe 'Hot-Spot': 0.0000
Classe 'Hot-Spot-Multi': 0.2500
Classe 'No-Anomaly': 0.7596
Classe 'Offline-Module': 0.0000
Classe 'Shadowing': 0.5570
Classe 'Soiling': 0.7500
Classe 'Vegetation': 0.5026

```

```

1 class_counts_val = np.bincount(val_data.targets)
2 weighted_avg_precision_vdd = weighted_precision(cm_arch_vdd, class_counts_val)
3 print(f"\nMédia ponderada da precisão (ponderada pela quantidade de amostras): {weighted_avg_precision_vdd:.4f}")

```

→ Média ponderada da precisão (ponderada pela quantidade de amostras): 0.5970

```

1 # Extract the names of all layers from the model.
2 all_layer_names = [name for name, layer in model._very_deep.named_modules() if name]
3
4 # Create a list of layers to visualize, EXCLUDING 'drop' AND 'pool'.
5 # This is the line that fixes the error.
6 layers_to_visualize = [name for name in all_layer_names if 'drop' not in name and 'pool' not in name]
7
8 print(f"Layers to be visualized: {layers_to_visualize}")
9 print("-" * 30)
10
11 try:

```

```

12 # Set the model to evaluation mode
13 arch_very_deep.model.eval()
14
15 # Attach hooks using the corrected, filtered list
16 arch_very_deep.attach_hooks(layers_to_hook=layers_to_visualize)
17 print("Hooks attached successfully.")
18
19 # Get a batch of images
20 torch.manual_seed(42)
21 images, labels = next(iter(val_loader))
22 images = images.to(arch_very_deep.device)
23
24 # Pass the images through the model
25 output = arch_very_deep.model(images)
26 print("Data propagated through the model.")
27
28 # Visualize the outputs
29 print("Generating visualization of feature maps...")
30 fig = arch_very_deep.visualize_outputs(layers=layers_to_visualize)
31 plt.show()
32 print("Visualization generated.")
33
34 # Remove the hooks
35 arch_very_deep.remove_hooks()
36 print("Hooks removed successfully.")
37
38 except Exception as e:
39     print(f"\nAn unexpected error occurred: {e}")

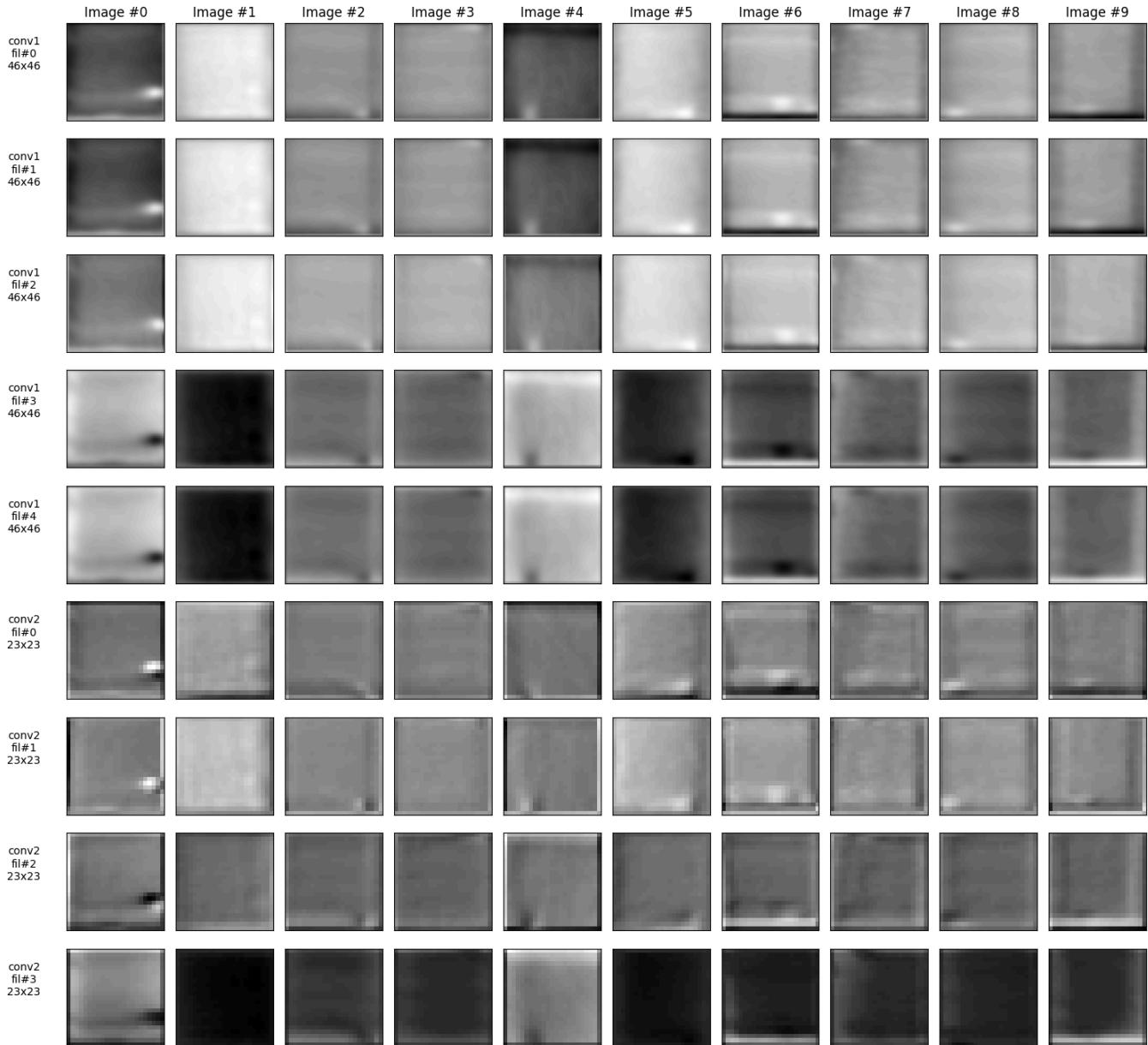
```

Layers to be visualized: ['conv1', 'conv2', 'conv3', 'conv4', 'fc1', 'fc2']

Hooks attached successfully.

Data propagated through the model.

Generating visualization of feature maps...

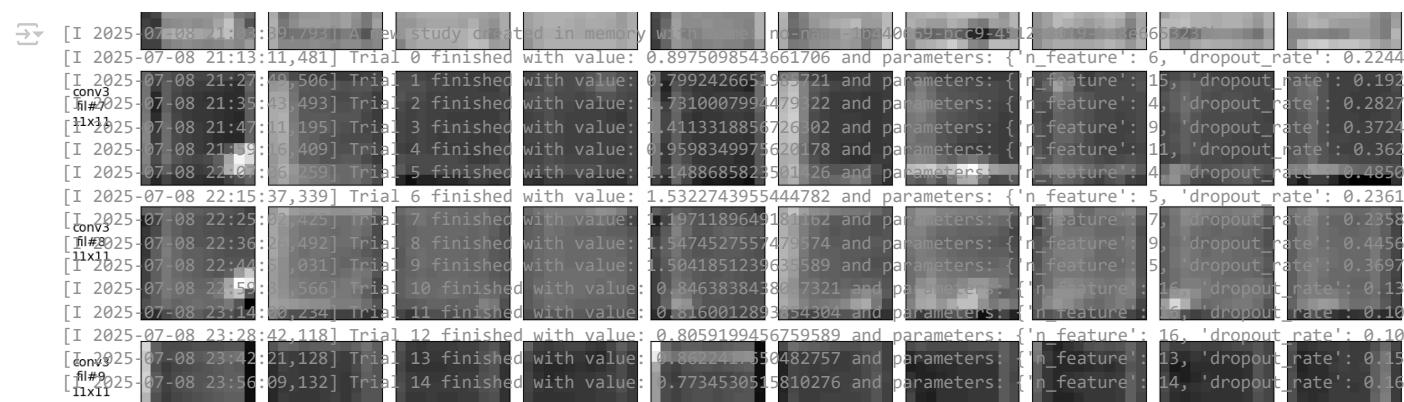




```

1 # Set a new, descriptive name for this experiment
2 mlflow.set_experiment("Deeper Network Optimization (CNN vs VeryDeepCNN")
3
4 def get_predictions(architecture, data_loader):
5     all_preds = []
6     all_labels = []
7     architecture.model.eval()
8     with torch.no_grad():
9         for images, labels in data_loader:
10             images = images.to(architecture.device)
11             outputs = architecture.model(images)
12             _, preds = torch.max(outputs, 1)
13             all_preds.extend(preds.cpu().numpy())
14             all_labels.extend(labels.cpu().numpy())
15     return all_preds, all_labels
16
17 def objective_deep(trial):
18     with mlflow.start_run():
19         params = {
20             'n_feature': trial.suggest_int('n_feature', 4, 16),
21             'dropout_rate': trial.suggest_float('dropout_rate', 0.1, 0.5),
22             'lr': trial.suggest_float('lr', 1e-5, 1e-3, log=True)
23         }
24     mlflow.log_params(params)
25
26     model = VeryDeepCNN(n_feature=params['n_feature'], p=params['dropout_rate'])
27
28     optimizer = optim.Adam(model.parameters(), lr=params['lr'])
29     arch = Architecture(model, multi_loss_fn, optimizer)
30     arch.set_loaders(train_loader, val_loader)
31
32     n_epochs = 15
33     arch.train(n_epochs)
34
35     # The rest of the function for logging metrics and artifacts remains exactly the same.
36     for epoch in range(n_epochs):
37         mlflow.log_metric('train_loss', arch.losses[epoch], step=epoch)
38         mlflow.log_metric('val_loss', arch.val_losses[epoch], step=epoch)
39
40     all_preds, all_labels = get_predictions(arch, val_loader)
41     class_names = val_data.classes
42     cm = confusion_matrix(all_labels, all_preds)
43     plt.figure(figsize=(12, 10))
44     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
45     plt.title('Confusion Matrix', fontsize=16)
46     plt.ylabel('True Label')
47     plt.xlabel('Predicted Label')
48     plt.xticks(rotation=45, ha='right')
49     plt.tight_layout()
50     plt.savefig("confusion_matrix.png")
51     mlflow.log_artifact("confusion_matrix.png", "plots")
52     plt.close()
53
54     final_val_loss = arch.val_losses[-1]
55     mlflow.log_metric('final_val_loss', final_val_loss)
56
57     return final_val_loss

```



```

1 study = optuna.create_study(direction='minimize')
2 study.optimize(objective_deep, n_trials=20) # 20 models test, for exemple
3
4 print("\nEstudo de otimização finalizado!")

```

```

[2025-07-08 21:13:19,793] [I] A new study created in memory with name 'n_feature-1by40-n9-0cc9-4512' and study_id 65824
[2025-07-08 21:13:19,481] [I] Trial 0 finished with value: 0.8975098543661706 and parameters: {'n_feature': 6, 'dropout_rate': 0.2244}
[2025-07-08 21:27:19,506] [I] Trial 1 finished with value: 0.799242665193721 and parameters: {'n_feature': 15, 'dropout_rate': 0.1929}
[2025-07-08 21:33:13,493] [I] Trial 2 finished with value: 1.73100799447922 and parameters: {'n_feature': 4, 'dropout_rate': 0.2827}
[2025-07-08 21:47:11,195] [I] Trial 3 finished with value: 1.4113318856726402 and parameters: {'n_feature': 9, 'dropout_rate': 0.3724}
[2025-07-08 21:53:16,409] [I] Trial 4 finished with value: 0.9598349975620178 and parameters: {'n_feature': 11, 'dropout_rate': 0.3625}
[2025-07-08 22:09:16,259] [I] Trial 5 finished with value: 1.148868582350126 and parameters: {'n_feature': 4, 'dropout_rate': 0.4856}
[2025-07-08 22:15:37,339] [I] Trial 6 finished with value: 1.5322743955444782 and parameters: {'n_feature': 5, 'dropout_rate': 0.2361}
[2025-07-08 22:25:24,425] [I] Trial 7 finished with value: 1.197118964918162 and parameters: {'n_feature': 7, 'dropout_rate': 0.2358}
[2025-07-08 22:36:10,492] [I] Trial 8 finished with value: 1.5474527557470574 and parameters: {'n_feature': 9, 'dropout_rate': 0.4456}
[2025-07-08 22:44:0,031] [I] Trial 9 finished with value: 1.504185123963589 and parameters: {'n_feature': 5, 'dropout_rate': 0.3697}
[2025-07-08 22:55:27,566] [I] Trial 10 finished with value: 0.846383843807321 and parameters: {'n_feature': 16, 'dropout_rate': 0.136}
[2025-07-08 23:14:10,234] [I] Trial 11 finished with value: 0.8160012893034304 and parameters: {'n_feature': 6, 'dropout_rate': 0.106}
[2025-07-08 23:28:42,118] [I] Trial 12 finished with value: 0.8059199456759589 and parameters: {'n_feature': 16, 'dropout_rate': 0.106}
[2025-07-08 23:42:21,128] [I] Trial 13 finished with value: 0.8362241350482757 and parameters: {'n_feature': 13, 'dropout_rate': 0.157}
[2025-07-08 23:56:09,132] [I] Trial 14 finished with value: 0.7734530515810276 and parameters: {'n_feature': 14, 'dropout_rate': 0.167}

```

```
[I 2025-07-09 00:10:41,376]: Trial 15 finished with value: 0.9739238861551944 and parameters: {'n_feature': 13, 'dropout_rate': 0.184}
[I 2025-07-09 00:23:25,738]: Trial 16 finished with value: 0.9502008390077885 and parameters: {'n_feature': 14, 'dropout_rate': 0.202}
[I 2025-07-09 00:37:09,190]: Trial 17 finished with value: 0.860909645346568 and parameters: {'n_feature': 14, 'dropout_rate': 0.183}
[T 2025-07-09 00:48:50,000]: Trial 18 finished with value: 0.816025124867901 and parameters: {'n_feature': 11, 'dropout_rate': 0.287}
[! 2025-07-09 01:02:17,497]: Trial 19 finished with value: 1.064749570124974 and parameters: {'n_feature': 14, 'dropout_rate': 0.205}
[xx#18
[xx#11
Estudo de otimização finalizado!
```

```
1 experiment_name = "Deeper Network Optimization (CNN vs VeryDeepCNN)"
2
3 print(f"Buscando resultados para o experimento: '{experiment_name}'")
4
5 try:
6     # Busca o experimento pelo nome
7     experiment = mlflow.get_experiment_by_name(experiment_name)
8
9     if experiment is None:
10         print(f"Erro: Experimento '{experiment_name}' não encontrado.")
11     else:
12         experiment_id = experiment.experiment_id
13
14     # Busca TODOS os runs (ensaios) do experimento
15     runs_df = mlflow.search_runs(experiment_ids=[experiment_id])
16
17     if runs_df.empty:
18         print("Nenhum run encontrado neste experimento.")
19     else:
20         print(f"Encontrados {len(runs_df)} resultados no experimento.")
21
22     # Salva o DataFrame completo em um arquivo CSV
23     csv_filename = "resultados_completos_otimizacao_mais_camadas.csv"
24     runs_df.to_csv(csv_filename, index=False)
25
26     print(f"\nResultados exportados com sucesso para o arquivo: '{csv_filename}'")
27
28     # Exibe as primeiras linhas da tabela de resultados
29     print("\nAmostra dos resultados:")
30     display(runs_df.head())
31
32 except Exception as e:
33     print(f"\nOcorreu um erro inesperado: {e}")
34     print("Verifique se o nome do experimento está correto e se o estudo já foi executado.")
```

Buscando resultados para o experimento: 'Deeper Network Optimization (CNN vs VeryDeepCNN)'
Encontrados 21 resultados no experimento.
Resultados exportados com sucesso para o arquivo: 'resultados_completos_otimizacao_mais_camadas.csv'
Amostra dos resultados:

	run_id	experiment_id	status	artifact_uri	start_time
0	fcbf4c4aa3a946ed8414e2bfe101f2d8	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/fcbf...	2025-07-09 00:48:59.610000+00:00
1	e4077de4d9264ad28f20bbec2a1ba	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/e407...	2025-07-09 00:37:09.194000+00:00
2	846d4bc2fb824331940262d9de842e40	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/846d...	2025-07-09 00:23:25.742000+00:00
3	fb164685b324689aa46316ba89532115556419399938459	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/fb16...	2025-07-09 00:00:14.350000+00:00
4	dbbc77b8a35c416e8fe54b6dc200b040	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/dbbc...	2025-07-08 23:56:09.136000+00:00

```
1 #Carregar os Dados do Arquivo CSV
2
3 try:
4     # Certifique-se de que o arquivo CSV está no mesmo ambiente
5     df_results = pd.read_csv("resultados_completos_otimizacao_mais_camadas.csv")
6     print("✅ Arquivo CSV carregado com sucesso!")
7 except FileNotFoundError:
8     print("❌ Erro: Arquivo 'resultados_completos_otimizacao.csv' não encontrado.")
9     # Se der este erro, faça o upload do arquivo para o Colab.
10    df_results = None
11
12 # Exibe as primeiras linhas da tabela de resultados
13 print("\nAmostra dos resultados:")
14 display(df_results)
```

	inv4 #5 5x5	Arquivo CSV carregado com sucesso!	run_id	experiment_id	status			artifact_uri	start_time
Amostra dos resultados:									
0	fcbf4c4aa3a946ed8414e2bfe101f2d8	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/fcbf...	2025-07-09 00:48:59.610000+00:00				
1	e4077de...02845bd981205bce2b1a3ba	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/e407...	2025-07-09 00:37:39.194000+00:00				
2	846d4bc2fb824331940262d9de842e40	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/846d...	2025-07-09 00:23:25.742000+00:00				
3	fb1...68...bb324e8...aa463076ba33943	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/bf1...	2025-07-09 00:09:41.380000+00:00				
4	dbbc77b8a35c416e8fe54b6dc200b040	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/dbbc...	2025-07-08 23:56:09.136000+00:00				
5	cb1de2fc81e9...e687835721143bf8b0	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/cb1d...	2025-07-08 23:42:51.133000+00:00				
6	be93ccd84c784e0ea8aa77c208b3efcb	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/be93...	2025-07-08 23:28:42.132000+00:00				
7	318bb78...971...cd9888d1bcb...5b6c6c	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/318b...	2025-07-08 23:14:10.238000+00:00				
8	7d04976717a047238e580e44e10b88e1	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/7d04...	2025-07-08 22:59:33.570000+00:00				
9	7...47c1b3f9d4f...4...621...fa1b4f52b	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/75a...	2025-07-08 22:44:58.160000+00:00				
10	5766adf1c2d9457cb5ef490c2732dcca	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/5766...	2025-07-08 22:36:26.497000+00:00				
11	c0df7...5a21ad4e5881d...c0fa6b27be3...	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/c0df...	2025-07-08 22:20:24.249000+00:00				
12	6dde7754538f46d29a810c37814be660	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/6dde...	2025-07-08 22:15:37.344000+00:00				
13	47ca981bd...642...ab76705963...f76617	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/47ca...	2025-07-08 22:07:06.264000+00:00				
14	59ee032133fd40b0bf005d2222ca05e	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/59ee...	2025-07-08 21:59:16.413000+00:00				
15	b7fc8bccb06f495c89ea536d105cd1d5	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/b7fc...	2025-07-08 21:47:11.202000+00:00				
16	c556c95a294d426e989e95a7a3ddcf	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/c556...	2025-07-08 21:35:43.497000+00:00				
17	conv4 fil#14 5x5 da4c3f02b00274466a344b1b4d...cc45c9	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/da4c...	2025-07-08 21:27:49.518000+00:00				
18	511311558f234b23b791d00f18426c45	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/5113...	2025-07-08 21:13:11.486000+00:00				
Melhor modelo:	conv4 fil#14 5x5 880dddea59...40b...a8b12984...30a425	185552419399938459	FINISHED	file:///content/mlruns/185552419399938459/880d...	2025-07-08 21:03:39.500000+00:00				

```

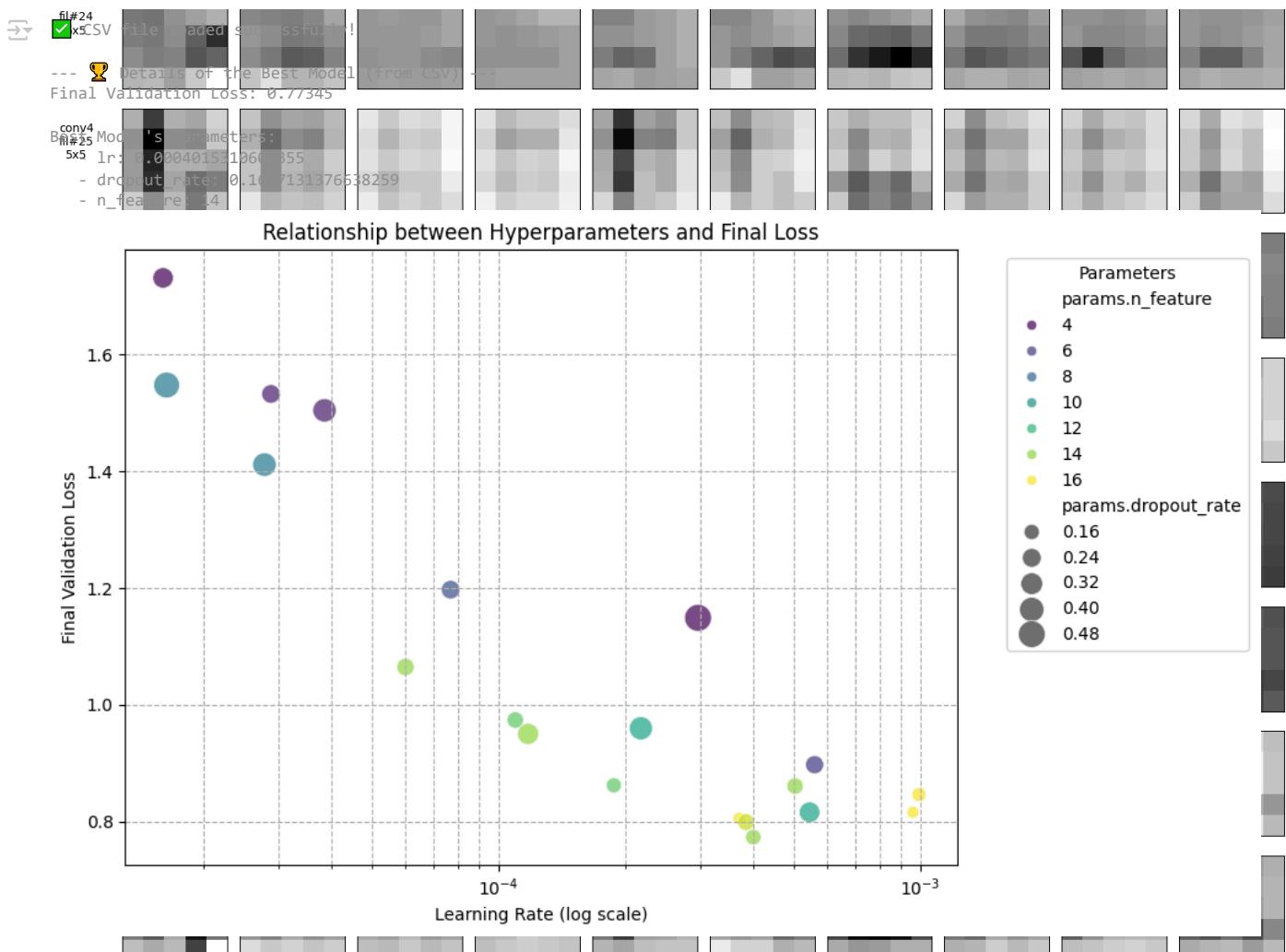
1 try:
2     # Make sure the CSV file is in the same environment
3     df_results = pd.read_csv("resultados_completos_otimizacao_mais_camadas.csv")
4     print("✅ CSV file loaded successfully!")
5 except FileNotFoundError:
6     print("❌ Error: 'resultados_completos_otimizacao.csv' file not found.")
7     # If this error occurs, upload the file to your Colab environment.
8     df_results = None
9
10 if df_results is not None:
11     # Sorts the DataFrame by the lowest final validation loss
12     best_run_from_csv = df_results.sort_values(by="metrics.final_val_loss").iloc[0]
13
14     print("\n--- 🎉 Details of the Best Model (from CSV) ---")
15     print(f"Final Validation Loss: {best_run_from_csv['metrics.final_val_loss']:.5f}")
16
17     print("\nBest Model's Parameters:")
18     # Filter columns to get only those starting with 'params.'
19     param_cols = [col for col in df_results.columns if col.startswith('params.')]
20     for param in param_cols:
21         param_name = param.replace('params.', '')
22         param_value = best_run_from_csv[param]
23         print(f" - {param_name}: {param_value}")
24
25     # Create a scatter plot to see the relationship between learning rate and final loss
26     plt.figure(figsize=(10, 6))

```

```

27     sns.scatterplot(
28         data=df_results,
29         x="params.lr",
30         y="metrics.final_val_loss",
31         hue="params.n_feature",      # Color by number of features
32         size="params.dropout_rate",  # Size by dropout rate
33         sizes=(50, 250),
34         palette='viridis',
35         alpha=0.7
36     )
37     plt.title("Relationship between Hyperparameters and Final Loss")
38     plt.xlabel("Learning Rate (log scale)")
39     plt.ylabel("Final Validation Loss")
40     plt.xscale('log')
41     plt.grid(True, which='both', linestyle='--')
42     plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', title='Parameters')
43     plt.tight_layout()
44     plt.show()
45

```



```

1 experiment_name = "Deeper Network Optimization (CNN vs VeryDeepCNN)"
2
3 # Busca todos os runs do nosso experimento, ordenando pelo menor 'final_val_loss'
4 best_run = mlflow.search_runs(
5     experiment_names=[experiment_name],
6     order_by=["metrics.final_val_loss ASC"],
7     max_results=1
8 )
9
10 if best_run.empty:
11     print(f"Nenhum run encontrado para o experimento: '{experiment_name}'. Execute a otimização primeiro.")
12 else:
13     # Pega o ID do melhor run
14     best_run_id = best_run.loc[0, 'run_id']
15     print(f"🏆 Melhor Run Encontrado! ID: {best_run_id}")
16     print(f" - Loss de Validação Final: {best_run.loc[0, 'metrics.final_val_loss']:.5f}")
17
18     # Imprimir os Hiperparâmetros do Melhor Modelo
19     print("\nParâmetros do Melhor Modelo (para recriá-lo):")
20     # Filtra as colunas para pegar apenas as que começam com 'params.'
21     best_params = {col.replace('params.', ''): best_run.loc[0, col] for col in best_run.columns if col.startswith('params.')}

```

```
22     for param_name, param_value in best_params.items():
23         # Formata o valor para ser mais legível
24         if isinstance(param_value, float):
25             print(f"    - {param_name}: {param_value:.6f}")
26         else:
27             print(f"    - {param_name}: {param_value}")
28
29     # Para buscar as métricas de cada época, precisamos do cliente do MLflow
30     client = mlflow.tracking.MlflowClient()
31
32     # Busca o histórico de métricas para o nosso melhor run
33     train_loss_history = client.get_metric_history(best_run_id, "train_loss")
34     val_loss_history = client.get_metric_history(best_run_id, "val_loss")
35
36     # Extrai os valores para plotagem
37     train_losses = [m.value for m in train_loss_history]
38     val_losses = [m.value for m in val_loss_history]
39
40     plt.figure(figsize=(14, 6))
41     plt.subplot(1, 2, 1)
42     plt.plot(train_losses, label='Loss de Treinamento', color='blue')
43     plt.plot(val_losses, label='Loss de Validação', color='red', linestyle='--')
44     plt.title('Curvas de Perda do Melhor Modelo', fontsize=14)
45     plt.xlabel('Épocas')
46     plt.ylabel('Loss')
47     plt.legend()
48     plt.grid(True)
49
50     # Baixa o artefato (a imagem da matriz de confusão) do melhor run
51     try:
52         # Define o caminho local para onde o artefato será baixado
53         local_path = client.download_artifacts(run_id=best_run_id, path="plots/confusion_matrix.png")
54
55         # Carrega e exibe a imagem
56         img = mpimg.imread(local_path)
57
58         plt.subplot(1, 2, 2)
59         plt.imshow(img)
60         plt.title('Matriz de Confusão do Melhor Modelo', fontsize=14)
61         plt.axis('off')
62
63     except Exception as e:
64         print(f"\nNão foi possível carregar a matriz de confusão: {e}")
65
66     plt.tight_layout()
67     plt.show()
```

➡️ 🏆 Melhor Run Encontrado! ID: cb1de2fc81e94ef6878957211a3bf8b0
- Loss de Validação Final: 0.77345

Parâmetros do Melhor Modelo (para recriá-lo):

- Tr. A 00001011000000000000000000000000

```
1 print("---- Starting Generation of the Full Report for All Runs ----")
2
3 try:
4     # Fetch ALL runs from the correct experiment for the deeper network
5     all_runs_df = mlflow.search_runs(
6         experiment_names=["Deeper Network Optimization (CNN vs VeryDeepCNN)"], # MODIFIED
7         # Sort so the best models appear first in the report
8         order_by=["metrics.final_val_loss ASC"]
9     )
10
11    if all_runs_df.empty:
12        print("\nNo runs found. Please run the optimization first.\n")
```