Instruções

objetivo geral: criar uma rede neural para classificação de imagens do dataset fashion mnist e exploração de todas as características da rede neural, uma por vez.

O melhor resultado de cada questão/etapa é utilizado nas próximas

QUESTÃO 01: exploração inicial

- Dataset fashion mnist
- função de ativação
- 5 testes com inicialização aleatória: diferenças de convergencia, estabilidade e desempenho
- dataset de treino
- métricas: medida de desempenho(accuracy), função de perda (entropia cruzada/loss), curva de convergência
- otimizador: Adam
- arquitetura: quantas camadas e neurônios por camada
- funcoes de ativação: ReLU, Sigmoid ou Tanh
- quantas épocas
- taxa de aprendizado
- indícios de under/overfitting

QUESTÃO 02: exploração de hiperparâmetros

- taxa de aprendizado x termo momento x velocidade de convergência
- Grid search para encontrar a melhor combinação: erro de treinamento x taxa de aprendizado x momento
- taxa de aprendizado menor e momento intermediário
- dataset de treino e (opcionalmente) dataset de validação
- métricas: função de perda, velocidade de convergência, curva de convergência e (opcional) estabilidade
- critério de parada
- combinação com melhor equilíbrio entre velocidadee estabilidade
- tendências observadas(ex: maior taxa de aprendizado leva a maior velocidade, mas menor estabilidade)

QUESTÃO 03: topologia de rede neural

- dataset de treino e (opcionalmente) dataset de validação
- impacto do número de camadas ocultas e neurônios por camada e teste de variação desses números
- métricas: função de perda, curva de convergência(under e overfitting), tempo de treinamento, generalização(medida F), precisão, revocação
- gráfico de perda mostrando diferença entre topologias

QUESTÃO 04: qualidade dos dados

- influência do número e qualidade dos dados, ruído, etc sobre a capacidade de generalização
- dividir o dataset em subsets de acordo com o rótulo -> manter proporcionalidade
- faixas do dataset: 10%, 30%, 50%, 70%, 100%
- métrica: função de perda, acurácia,
- identificar saturação no aprendizado
- curvas de generalização: tamanho do conjunto X desempenho
- tempo de treinamento e custo computacional
- Estratégia de amostragem(estratificada, aleatória ou outra)

QUESTÃO 05:

- escolher 4 melhores modelos e usar modelo de testes neles
- treinamento como referência comparativa
- ajustes de otimização
- métricas: perda(entropia cruzada categórica), acurácia, curva de validação(treinamento x teste), F1 score, precisão, revocação
- escolha da configuração final do modelo

QUESTÃO 06: validação cruzada k-fold

divisão do dataset em k-subconjuntos e teste em todos eles

- métricas: média de todas as partições de perda: acurácia e F1.
- para cada partição: curvas de validação e variância(dispersão) dos resultados
- justificativa do tamanho de k
- identificação de flutuações

```
import numpy as np
import matplotlib.pyplot as plt
#from tensorflow import kerasimpor
from tensorflow import keras
from sklearn.model_selection import train_test_split
import secrets
import pickle
from pathlib import Path
import time
import gc
from tensorflow.keras.callbacks import EarlyStopping
```

Divisão do dataset

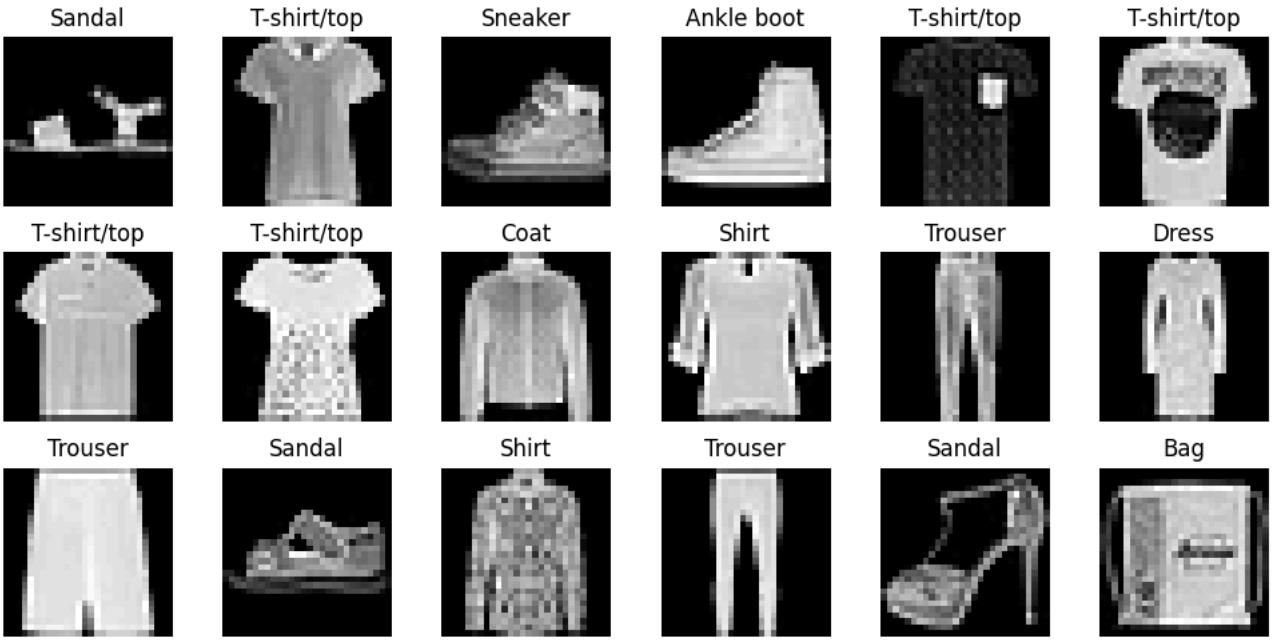
```
#dataset já dividido em treino e teste
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()
#split de treino entre 80% treino e 20% validação
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=42, stratify=y_train
# Normalização (0-1) para visualização e futura modelagem
x_train = x_train.astype("float32")/255.0
x_test = x_test.astype("float32")/255.0
x_val = x_val.astype("float32")/255.0
"""
converte inteiro discreto de 0 a 255 para contínuo float de 0.0 a 1.0
redes neurais funcionam melhor com entradas contínuas e escala pequena e próxima
float representa melhor valores intermediários entre 2 as cores possíveis (preto e branco)
y é inteiro de 0 a 9, sendo o número sua classe, não precisa de normalização
"""
```

```
'\nconverte inteiro discreto de 0 a 255 para contínuo float de 0.0 a 1.0\nredes neurais funcionam melhor com entradas
contínuas e escala pequena e próxima\nfloat representa melhor valores intermediários entre 2 as cores possíveis
(preto e branco)\ny é inteiro de 0 a 9, sendo o número sua classe, não precisa de normalização\n'
```

vizualização do dataset Fashion-MNIST

```
labels = [
    "T-shirt/top","Trouser","Pullover","Dress","Coat","Sandal","Shirt","Sneaker","Bag","Ankle boot"
]

print(f"Treino: {x_train.shape}, Validação: {x_val.shape}, Teste: {x_test.shape}")
print("Exemplo de rótulos (0-9):", labels)

# Grid de amostras aleatórias do conjunto de treino
fig, axes = plt.subplots(3, 6, figsize=(10, 5))
for i, ax in enumerate(axes.ravel()):
    idx = np.random.randint(0, len(x_train))
    ax.imshow(x_train[idx], cmap="gray")
    ax.set_title(labels[y_train[idx]])
    ax.axis("off")
plt.tight_layout()
plt.show()
```

```
Treino: (48000, 28, 28), Validação: (12000, 28, 28), Teste: (10000, 28, 28)
Exemplo de rótulos (0-9): ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag'
```



Modelo

```
"""
configuração padrão:
    camada de entrada com 784 neurônios(cada pixel da imagem 28x28)
    2 camadas ocultas (64 e 32 neurônios)
    camada de saída com 10 neurônios (10 classes)
"""

def build_model(learning_rate=1e-3, beta1=0.9, activation_hidden = 'relu', activation_output = 'softmax', num_hidden_

    layers = [
        keras.layers.InputLayer(shape=(28, 28)),  # imagens 28x28 pixels, cada pixel é um neurônio de entrada
        keras.layers.Flatten()  # transforma matriz 2D 28x28 em vetor 1D com 784 elementos
    ]
    # adiciona dinamicamente as camadas ocultas conforme num_hidden_layers
    for i in range(num_hidden_layers):
        layers.append(keras.layers.Dense(neurons_per_layer[i], activation_hidden))  # ReLU como função de ativação não

    # camada de saída
    layers.append(keras.layers.Dense(10, activation_output))  # 10 saídas (classes) possíveis

    # modelo sequencial -> "clássico" com uma camada após a outra
    model = keras.Sequential(layers)

    optimizer = keras.optimizers.Adam(learning_rate, beta1)
    model.compile(
        optimizer = optimizer,  # aprendizado adaptativo
        loss='sparse_categorical_crossentropy',  # ideal para classificação multiclasse com rótulos inteiros
        metrics=['accuracy']  # medida de desempenho simples
    )
    return model
```

gerador de seeds

```
PRIME_STEP = 2654435761  # grande e usado em hashing
MASK32 = 0xFFFFFFFF
base = secrets.randbits(32)

# ===== Método para "espaçar" mais as seeds =====
# Ideia: usar uma base aleatória de 32 bits e aplicar um incremento grande e primo
# (ex: 2654435761 = constante de Knuth) gerando progressão pseudo-dispersada em 32 bits.
# Depois aplicamos uma mistura (hash simples) para minimizar correlação linear.
```

```
def spaced_seeds(n, base_seed, step):
    seeds = []
    for i in range(n):
        raw = (base_seed + i * step) & MASK32
        # Mistura extra: multiplicação + xor + shift (barato, evita sequência muito próxima)
        mixed = (raw * 0x9E3779B1) & MASK32
        mixed ^= (mixed >> 16)
        seeds.append(mixed)
    return seeds

seeds = spaced_seeds(5, base, PRIME_STEP)
```

Checkpoints de treino

```
def load_checkpoint(checkpoint_file, q_name):
    if Path(checkpoint_file).is_file():
        print("\n✓ Carregando checkpoint anterior...")
        with open(checkpoint_file, 'rb') as f:
            checkpoint = pickle.load(f)
        results = checkpoint[f'results_{q_name}']
        histories = checkpoint[f'histories_{q_name}']
        start_combo = checkpoint['last_combination']
        print(f"  Retomando de {start_combo} combinações já processadas")
    else:
        results = []
        histories = []
        start_combo = 0
        print("criando arquivo de checkpoint: ", checkpoint_file)
    return results, histories, start_combo
```

```
def save_checkpoint(checkpoint_file, results, histories, current_combination, start_combo, total_combinations, check
    if current_combination % checkpoint_interval== 0 or current_combination == total_combinations:
        gc.collect()
        checkpoint_data = {
            'results': results,
            'histories': histories,
            'last_combination': current_combination
        }
        with open(checkpoint_file, 'wb') as f:
            pickle.dump(checkpoint_data, f)

        tempo_decorrido = (current_combination - start_combo) * 20 * np.mean([r['time_mean'] for r in results[-10:]]
        print(f"✓ Checkpoint #{current_combination // checkpoint_interval} | Progresso: {current_combination}/{total
```

```
def show_checkpoint(path, max_items=3):
    """Mostra um resumo rápido do checkpoint (Q2).
    - path: caminho para o .pkl
    - max_items: quantos itens do results_q2 mostrar
    """
    p = Path(path)
    if not p.exists():
        print(f"Arquivo não encontrado: {p}")
        return
    with open(p, 'rb') as f:
        print("✓ Carregando checkpoint no caminho:", p.resolve())
        data = pickle.load(f)
    keys = list(data.keys())
    print(f"✓ Checkpoint carregado")
    print(f"Campos: {keys}")
    print(f"Total de combinações salvas: {len(data.get('results', []))}")
    print(f"Total de históricos salvos: {len(data.get('histories', []))}")
    print(f"Última combinação: {data.get('last_combination')}")
    # Mostra amostra dos resultados
    sample = data.get('results', [])[:max_items]
    if sample:
        print(f"\nAmostra (até {max_items}) de results:")
        for i, r in enumerate(sample, 1):
            print(f"#{i}: epochs={r['epochs']}, lr={r['learning_rate']}, batch={r['batch_size']}, beta1={r['beta1']}
                  f"loss_mean={r['loss_mean']:.4f}, acc_mean={r['accuracy_mean']:.4f}, time_mean={r['time_mean']:.2f
    else:
        print("Nenhum resultado salvo no checkpoint.")
```

Questão 01: Rede neural simples

treinamento

```
# ========== CONFIGURAR CHECKPOINT PARA Q1 ==========
checkpoint_dir = Path('checkpoints')
checkpoint_dir.mkdir(exist_ok=True)
checkpoint_file_q1 = checkpoint_dir / 'results_q1_checkpoint.pkl'

# ========== CARREGAR CHECKPOINT SE EXISTIR ==========
histories_q1, final_metrics_q1, start_seed_idx = load_checkpoint(checkpoint_file_q1, 'q1')
log_lines = []

# Se checkpoint carregado, final_metrics tem dados
if final_metrics_q1:
    print(f"✓ Carregado {len(final_metrics_q1)} execuções do checkpoint anterior")
    seed_start = len(final_metrics_q1)
else:
    seed_start = 0
    final_metrics_q1 = []
    histories_q1 = []

# ========== TREINAMENTO Q1 COM CHECKPOINT ==========
for i, seed in enumerate(seeds[seed_start:], start=seed_start+1):
    keras.utils.set_random_seed(seed)
    model = build_model()

    h = model.fit(
        x_train, y_train,
        epochs=5,
        batch_size=128,
        verbose=0
    )
    histories_q1.append(h)

    final_metrics_q1.append({
        'run': i,
        'seed': seed,
        'final_train_loss': h.history['loss'][-1],
        'final_train_acc': h.history['accuracy'][-1]
    })

    log_lines.append(
        f"=== Treinamento {i}/5 (seed={seed}) ===\n"
        f"Train - Loss: {h.history['loss'][-1]:.4f}, accuracy: {h.history['accuracy'][-1]:.4f}"
    )

    # Salvar checkpoint a cada execução
    save_checkpoint(checkpoint_file_q1, final_metrics_q1, histories_q1, i, seed_start, len(seeds), q_name='q1', chec

    # Limpar memória
    del model
    keras.backend.clear_session()
    gc.collect()

print("\n".join(log_lines))
print(f"\n✓ Q1 concluído: {len(final_metrics_q1)} seeds treinadas")
```

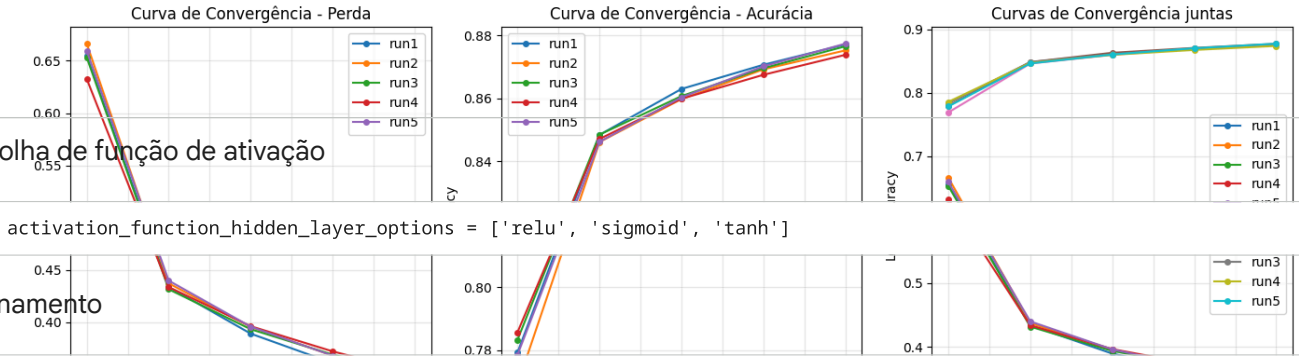
```
=== Treinamento 1/5 (seed=3824168193) ===
Train - Loss: 0.3406, accuracy: 0.8771
=== Treinamento 2/5 (seed=3822549125) ===
Train - Loss: 0.3475, accuracy: 0.8752
=== Treinamento 3/5 (seed=3820913677) ===
Train - Loss: 0.3509, accuracy: 0.8766
=== Treinamento 4/5 (seed=3819296689) ===
Train - Loss: 0.3521, accuracy: 0.8739
=== Treinamento 5/5 (seed=3817661433) ===
Train - Loss: 0.3473, accuracy: 0.8774
```

visualização

```
# ========== CURVAS DE CONVERGÊNCIA ==========
fig, axes = plt.subplots(1, 3, figsize=(14, 5))

print(f"estrutura das histories: {histories[-1].history}")
print("é possível adicionar mais informações no dicionário history, como f1, recall, precision, etc.")

#perda
for i, h in enumerate(histories, start=1):
    axes[0].plot(h.history['loss'], label=f'run{i}', marker='o', markersize=4)
axes[0].set_title('Curva de Convergência - Perda')
axes[0].set_xlabel('Época')
axes[0].set_ylabel('Loss')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

#acurácia
for i, h in enumerate(histories, start=1):
    axes[1].plot(h.history['accuracy'], label=f'run{i}', marker='o', markersize=4)
axes[1].set_title('Curva de Convergência - Acurácia')
axes[1].set_xlabel('Época')
axes[1].set_ylabel('Accuracy')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

#as duas(análise de over/underfitting)
for i, h in enumerate(histories, start=1):
    axes[2].plot(h.history['loss'], label=f'run{i}', marker='o', markersize=4)
for i, h in enumerate(histories, start=1):
    axes[2].plot(h.history['accuracy'], label=f'run{i}', marker='o', markersize=4)
axes[2].set_title('Curvas de Convergência juntas')
axes[2].set_xlabel('Época')
axes[2].set_ylabel('Loss/Accuracy')
axes[2].legend()
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
#loss continua alta, accuracy continua baixa -> underfitting
#loss continua caindo mesmo com accuracy estagnada -> overfitting
# ========== ESTABILIDADE ==========
train_losses = [m['final_train_loss'] for m in final_metrics]
train_accuracies = [m['final_train_acc'] for m in final_metrics]

print("\n========== ESTABILIDADE ==========")
print(f"Loss - média: {np.mean(train_losses):.4f}")
print(f"Loss - desvio padrão: {np.std(train_losses):.4f}")
print(f"accuracy  - média: {np.mean(train_accuracies):.4f}")
print(f"accuracy  - desvio padrão: {np.std(train_accuracies):.4f}")

fig, axes = plt.subplots(1, 2, figsize=(10, 4))
#5 seeds divididas entre bigode superior(máximo), limite superior da caixa, linha laranja (mediana), limite inferior
axes[0].boxplot(train_losses, whis=(0, 100))
axes[0].set_title('Estabilidade - Dispersão da Perda')
axes[0].set_ylabel('Loss')
axes[0].set_xticklabels(['Treino'])
#axes[0].scatter([1]*len(train_losses), train_losses, color='red', zorder=2)
axes[0].axhline(y=np.mean(train_losses), color='green', linestyle='--', linewidth=2, label='Média')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

axes[1].boxplot(train_accuracies, whis=(0, 100))
axes[1].set_title('Estabilidade - Dispersão da Acurácia')
axes[1].set_ylabel('Accuracy')
axes[1].set_xticklabels(['Treino'])
#axes[1].scatter([1]*len(train_accuracies), train_accuracies, color='red', zorder=2)
axes[1].axhline(y=np.mean(train_accuracies), color='green', linestyle='--', linewidth=2, label='Média')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# ========== DESEMPENHO ==========
print("\n========== DESEMPENHO por seed ==========")
for m in final_metrics:
    print(f"Run {m['run']} (seed={m['seed']}): Loss={m['final_train_loss']:.4f}, accuracy={m['final_train_acc']:.4f}
```

```
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
x = np.arange(1, 6)

axes[0].bar(x, train_losses, alpha=0.7, color='steelblue')
axes[0].set_title('Desempenho - Perda Final por Seed')
axes[0].set_xlabel('Run')
axes[0].set_ylabel('Loss')
axes[0].set_xticks(x)
axes[0].grid(True, alpha=0.3, axis='y')

axes[1].bar(x, train_accuracies, alpha=0.7, color='coral')
axes[1].set_title('Desempenho - Acurácia Final por Seed')
axes[1].set_xlabel('Run')
axes[1].set_ylabel('Accuracy')
axes[1].set_xticks(x)
axes[1].grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()

print("\nSeeds usadas:", seeds)
```

```
estrutura das histories: {'accuracy': [0.7785208225250244, 0.8462083339691162, 0.8603333234786987, 0.870187520980835
é possível adicionar mais informações no dicionário history, como f1, recall, precision, etc.
```



escolha de função de ativação

```
activation_function_hidden_layer_options = ['relu', 'sigmoid', 'tanh']
```

treinamento

```
# ========== TESTE DE FUNÇÕES DE ATIVAÇÃO COM CHECKPOINT ==========
seeds_q1_activation = spaced_seeds(20, base, PRIME_STEP)

# ========== DEFINIR CAMINHO DO CHECKPOINT ==========
checkpoint_dir = Path('checkpoints')
checkpoint_dir.mkdir(exist_ok=True)
checkpoint_file_q1_activation = checkpoint_dir / 'results_q1_activation_checkpoint.pkl'

print(f"Checkpoints serão salvos em: {checkpoint_file_q1_activation.absolute()}")

# ========== CARREGAR CHECKPOINT SE EXISTIR ==========
results_q1_activation, _, start_combo = load_checkpoint(checkpoint_file_q1_activation, 'q1_activation')

total_activations = len(activation_function_hidden_layer_options)
current_activation = 0

# ========== TREINAMENTO COM CHECKPOINT ==========
for activation_function_hidden_layer in activation_function_hidden_layer_options:
    current_activation += 1

    # Se já foi processado, pula
    if current_activation <= start_combo:
        print(f"⊘ Saltando {activation_function_hidden_layer} (já processada)")
        continue

    run_losses = []
    run_accuracies = []
    run_times = []

    print(f"\n{'='*60}")
    print(f"Testando função de ativação: {activation_function_hidden_layer}")
    print(f"{'='*60}")

    for seed_idx, s in enumerate(seeds_q1_activation, start=1):
        keras.utils.set_random_seed(s)
        model = build_model(activation_hidden=activation_function_hidden_layer)

        # Early stopping para acelerar
        early_stop = EarlyStopping(
            monitor='loss',
            patience=5,
            restore_best_weights=True,
            verbose=0
        )

        # Medir tempo
        start_time = time.time()
        h = model.fit(x_train, y_train, epochs=40, verbose=0, callbacks=[early_stop])
        training_time = time.time() - start_time

        run_losses.append(h.history['loss'][-1])
        run_accuracies.append(h.history['accuracy'][-1])
        run_times.append(training_time)

        # Limpar memória
        del model
        keras.backend.clear_session()

        # Feedback a cada 5 seeds
        if seed_idx % 5 == 0:
```

```
            print(f"  Progresso: {seed_idx}/20 seeds processadas...")

    results_q1_activation.append({
        'activation_function_hidden_layer': activation_function_hidden_layer,
        'loss_mean': float(np.mean(run_losses)),
        'loss_std': float(np.std(run_losses)),
        'accuracy_mean': float(np.mean(run_accuracies)),
        'accuracy_std': float(np.std(run_accuracies)),
        'time_mean': float(np.mean(run_times)),
        'time_std': float(np.std(run_times))
    })

    # ========== SALVAR CHECKPOINT ==========
    save_checkpoint(checkpoint_file_q1_activation, results_q1_activation, [], current_activation, start_combo, total

    print(f"✓ {activation_function_hidden_layer:12s} | Loss: {np.mean(run_losses):.4f}±{np.std(run_losses):.4f} | Ac
    gc.collect()

print(f"\n✓ Teste de funções de ativação concluído: {len(results_q1_activation)} funções testadas")
print(f"✓ Checkpoint final salvo em '{checkpoint_file_q1_activation.absolute()}'")
```

```
Checkpoints serão salvos em: c:\Users\User\Desktop\computação\S4\IC\trabalho 2 IC fashion MNIST\checkpoints\results_
criando arquivo de checkpoint:  checkpoints\results_q1_activation_checkpoint.pkl

============================================================
Testando função de ativação: relu
============================================================
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[13], line 48
     46 # Medir tempo
     47 start_time = time.time()
---> 48 h = model.fit(x_train, y_train, epochs=40, verbose=0, callbacks=[early_stop])
     49 training_time = time.time() - start_time
     51 run_losses.append(h.history['loss'][-1])

File c:\Users\User\Desktop\computação\S4\IC\trabalho 2 IC fashion MNIST\.venv\Lib\site-
packages\keras\src\utils\traceback_utils.py:117, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    115 filtered_tb = None
    116 try:
--> 117     return fn(*args, **kwargs)
    118 except Exception as e:
    119     filtered_tb = _process_traceback_frames(e.__traceback__)

File c:\Users\User\Desktop\computação\S4\IC\trabalho 2 IC fashion MNIST\.venv\Lib\site-
packages\keras\src\backend\tensorflow\trainer.py:399, in TensorFlowTrainer.fit(self, x, y, batch_size, epochs,
verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch,
steps_per_epoch, validation_steps, validation_batch_size, validation_freq)
    397 for begin_step, end_step, iterator in epoch_iterator:
    398     callbacks.on_train_batch_begin(begin_step)
--> 399     logs = self.train_function(iterator)
    400     callbacks.on_train_batch_end(end_step, logs)
    401     if self.stop_training:

File c:\Users\User\Desktop\computação\S4\IC\trabalho 2 IC fashion MNIST\.venv\Lib\site-
packages\keras\src\backend\tensorflow\trainer.py:242, in TensorFlowTrainer._make_function.
<locals>.function(iterator)
    238 if isinstance(
    239     iterator, (tf.data.Iterator, tf.distribute.DistributedIterator)
    240 ):
    241     opt_outputs = multi_step_on_iterator(iterator)
--> 242     if not opt_outputs.has_value():
    243         raise StopIteration
    244     return opt_outputs.get_value()

File c:\Users\User\Desktop\computação\S4\IC\trabalho 2 IC fashion MNIST\.venv\Lib\site-
packages\tensorflow\python\data\ops\optional_ops.py:176, in _OptionalImpl.has_value(self, name)
    174 def has_value(self, name=None):
    175   with ops.colocate_with(self._variant_tensor):
--> 176     return gen_optional_ops.optional_has_value(
    177         self._variant_tensor, name=name
    178     )

File c:\Users\User\Desktop\computação\S4\IC\trabalho 2 IC fashion MNIST\.venv\Lib\site-
packages\tensorflow\python\ops\gen_optional_ops.py:172, in optional_has_value(optional, name)
    170 if tld.is_eager:
    171   try:
--> 172     _result = pywrap_tfe.TFE_Py_FastPathExecute(
    173       _ctx, "OptionalHasValue", name, optional)
    174     return _result
```

ordenação

```
# Ordena por melhor equilíbrio: alta acurácia média, baixa perda média e baixa variância
# Score simples: accuracy_mean - loss_mean - (loss_std + accuracy_std)
sorted_results_q1 = sorted(
    results_q1,
    key=lambda sorted_result: (-(sorted_result['accuracy_mean']), sorted_result['loss_mean'], sorted_result['loss_st
)

print("Funções de ativação(melhor pra pior):")
for i,sorted_result in enumerate(sorted_results_q1[:3]):
    print(
        f"{i+1}. activation_function_hidden_layer={sorted_result['activation_function_hidden_layer']}"
        f" - Loss(média/desvio): {sorted_result['loss_mean']:.4f}/{sorted_result['loss_std']:.4f}, "
        f"Accuracy(média/desvio): {sorted_result['accuracy_mean']:.4f}/{sorted_result['accuracy_std']:.4f}"
    )
```

```
Funções de ativação(melhor pra pior):
1. activation_function_hidden_layer=sigmoid - Loss(média/desvio): 0.1402/0.0023, Accuracy(média/desvio): 0.9521/0.001
2. activation_function_hidden_layer=tanh - Loss(média/desvio): 0.1388/0.0038, Accuracy(média/desvio): 0.9499/0.0018
3. activation_function_hidden_layer=relu - Loss(média/desvio): 0.1415/0.0041, Accuracy(média/desvio): 0.9473/0.0016
```

Questão 02: hiperparâmetros

parâmetros ajustados

```
#TODO: mais opcões de hiperparâmetros para teste exaustivo final
# Dividir em múltiplas execuções para evitar timeout de 12 horas
# Execute uma variável por vez ou em pequenos lotes

num_epochs_grid = [5, 10, 20, 30, 40]
learning_rates = [1e-4, 1e-3, 1e-2, 1e-1]
batch_sizes = [32, 64, 128, 256]
momentums_beta1 = [0.5, 0.7, 0.9, 0.99]

print(f"Total de combinações a testar: {len(num_epochs_grid) * len(learning_rates) * len(batch_sizes) * len(momentums_
print(f"Tempo estimado: ~{len(num_epochs_grid) * len(learning_rates) * len(batch_sizes) * len(momentums_beta1) * 20 /
```

```
Total de combinações a testar: 320
Tempo estimado: ~1.8 horas (com 20 seeds)
```

treinamento

```
#TODO: aumentar número de seeds para teste exaustivo final
#TODO: treino e validação
import time
import gc
import pickle
from pathlib import Path

seeds_q2 = spaced_seeds(20, base, PRIME_STEP)

# ========== DEFINIR CAMINHO DO CHECKPOINT ==========
checkpoint_dir = Path('checkpoints')
checkpoint_dir.mkdir(exist_ok=True)
checkpoint_file = checkpoint_dir / 'results_q2_checkpoint.pkl'

print(f"Checkpoints serão salvos em: {checkpoint_file.absolute()}")

# ========== CARREGAR CHECKPOINT SE EXISTIR ==========
results_q2, histories_q2, start_combo = load_checkpoint(checkpoint_file, 'q2')

total_combinations = len(num_epochs_grid) * len(learning_rates) * len(batch_sizes) * len(momentums_beta1)
current_combination = 0

# ========== EARLY STOPPING COM CALLBACK ==========
from tensorflow.keras.callbacks import EarlyStopping

for epochs in num_epochs_grid:
```

```
    for learning_rate in learning_rates:
        for batch_size in batch_sizes:
            for beta1 in momentums_beta1:
                current_combination += 1

                # Pula combinações já processadas (não treina novamente)
                if current_combination <= start_combo:
                    continue

                run_losses = []
                run_accuracies = []
                run_times = []

                for s in seeds_q2:
                    keras.utils.set_random_seed(s)
                    model = build_model(learning_rate=learning_rate, beta1=beta1)

                    # Early stopping: interrompe se não houver melhoria por 5 épocas
                    early_stop = EarlyStopping(
                        monitor='loss',
                        patience=5,
                        restore_best_weights=True,
                        verbose=0
                    )

                    # Mede tempo de treinamento
                    start_time = time.time()
                    h = model.fit(
                        x_train, y_train,
                        epochs=epochs,
                        batch_size=batch_size,
                        callbacks=[early_stop],
                        verbose=0
                    )
                    training_time = time.time() - start_time

                    histories_q2.append(h)
                    run_losses.append(h.history['loss'][-1])
                    run_accuracies.append(h.history['accuracy'][-1])
                    run_times.append(training_time)

                    # Limpa modelo para liberar memória
                    del model
                    keras.backend.clear_session()

                results_q2.append({
                    'epochs': epochs,
                    'learning_rate': learning_rate,
                    'batch_size': batch_size,
                    'beta1': beta1,
                    'loss_mean': float(np.mean(run_losses)),
                    'loss_std': float(np.std(run_losses)),
                    'accuracy_mean': float(np.mean(run_accuracies)),
                    'accuracy_std': float(np.std(run_accuracies)),
                    'time_mean': float(np.mean(run_times)),
                    'time_std': float(np.std(run_times))
                })

                # ========== SALVAR CHECKPOINT ==========
                save_checkpoint(checkpoint_file, results_q2, histories_q2, current_combination, start_combo, total_com

print(f"\n✓ Treinamento Q2 concluído: {len(results_q2)} combinações testadas")
print(f"✓ Checkpoint final salvo em '{checkpoint_file.absolute()}'")

"""Remover arquivo de checkpoint após conclusão bem-sucedida
if checkpoint_file_q2.exists():
    checkpoint_file_q2.unlink()
    print("✓ Arquivo de checkpoint removido (conclusão bem-sucedida)")"""
```

Show hidden output

```
show_checkpoint(checkpoint_dir / 'results_q2_checkpoint_2.pkl', max_items=5)
```

```
✓ Carregando checkpoint no caminho: C:\Users\User\Desktop\computação\S4\IC\trabalho 2 IC fashion MNIST\checkpoints\re
✓ Checkpoint carregado
Campos: ['results_q2', 'histories_q2', 'last_combination']
Total de combinações salvas: 0
```

```
Total de históricos salvos: 0
Última combinação: 125
Nenhum resultado salvo no checkpoint.
```

```
#resultados e histórico finais
results_q2 = []
histories_q2 = []
with open(checkpoint_dir / 'results_q2_checkpoint.pkl', 'rb') as f:
    checkpoint = pickle.load(f)
results_q2 = checkpoint['results_q2']
histories_q2 = checkpoint['histories_q2']
print("resultados do primeiro treino coletados")
with open(checkpoint_dir / 'results_q2_checkpoint_2.pkl', 'rb') as f:
    checkpoint = pickle.load(f)
results_q2 += checkpoint['results_q2']
histories_q2 += checkpoint['histories_q2']
print("resultados do segundo treino coletados")
with open(checkpoint_dir / 'results_q2_checkpoint_3.pkl', 'rb') as f:
    checkpoint = pickle.load(f)
results_q2 += checkpoint['results_q2']
histories_q2 += checkpoint['histories_q2']

print("tamanho de results_q2: ", len(results_q2), " tamanho de histories_q2: ", len(histories_q2))
```

```
resultados do primeiro treino coletados
resultados do segundo treino coletados
tamanho de results_q2:  317  tamanho de histories_q2:  6340
```

ordenação

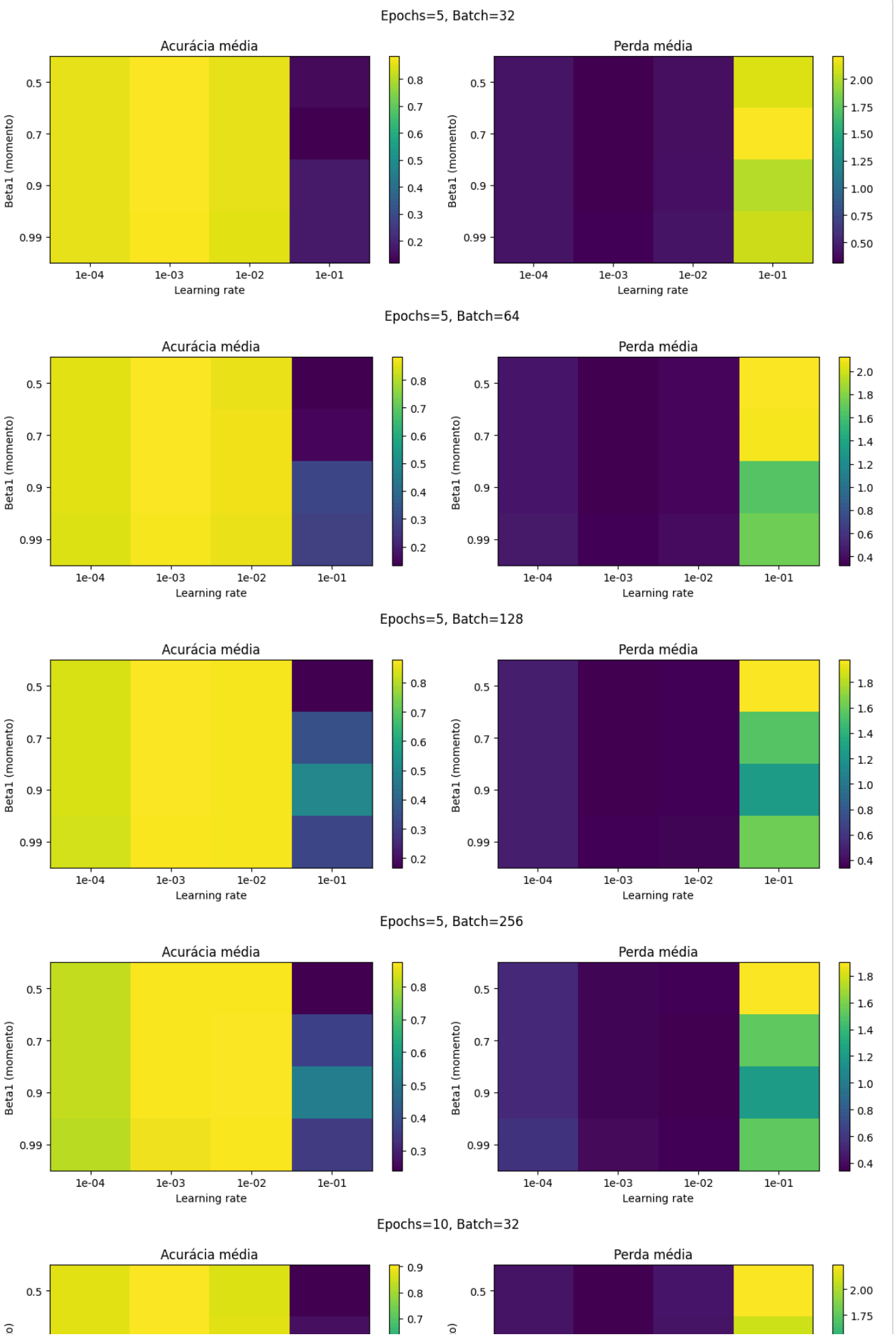
```
# Ordena por melhor equilíbrio: alta acurácia média, baixa perda média e baixa variância
# Score simples: accuracy_mean - loss_mean - (loss_std + accuracy_std)
sorted_results_q2 = sorted(
    results_q2,
    key=lambda sorted_result: (-(sorted_result['accuracy_mean']), sorted_result['loss_mean'], sorted_result['loss_st
)

print("Top 10 melhores combinações (melhor pro pior):")
for i,sorted_result in enumerate(sorted_results_q2[:10]):
    print(
        f"{i+1}. epochs={sorted_result['epochs']}, learning_rate={sorted_result['learning_rate']},"
        f" batch={sorted_result['batch_size']}, beta1={sorted_result['beta1']} | "
        f"loss_mean={sorted_result['loss_mean']:.4f} (±{sorted_result['loss_std']:.4f}), "
        f"accuracy_mean={sorted_result['accuracy_mean']:.4f} (±{sorted_result['accuracy_std']:.4f})"
    )

print("\n\nTop 10 piores combinações (melhor pro pior):")
for i,sorted_result in enumerate(sorted_results_q2[-10:-1]):
    print(
        f"{i+1}. epochs={sorted_result['epochs']}, learning_rate={sorted_result['learning_rate']},"
        f" batch={sorted_result['batch_size']}, beta1={sorted_result['beta1']} | "
        f"loss_mean={sorted_result['loss_mean']:.4f} (±{sorted_result['loss_std']:.4f}), "
        f"accuracy_mean={sorted_result['accuracy_mean']:.4f} (±{sorted_result['accuracy_std']:.4f})"
    )
```

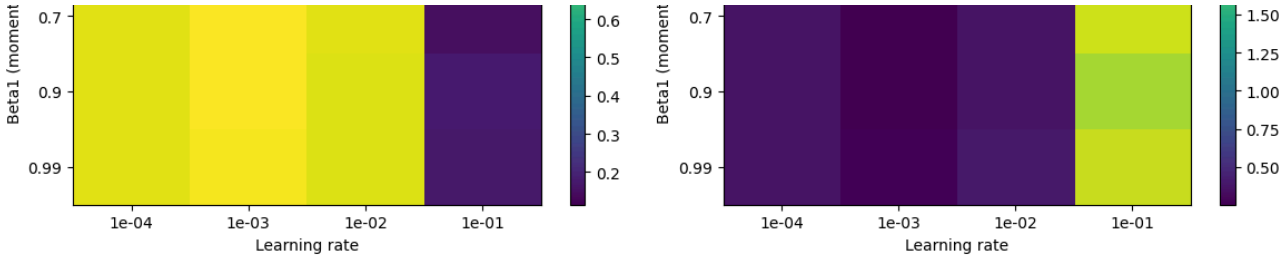
```
Top 10 melhores combinações (melhor pro pior):
1. epochs=40, learning_rate=0.001, batch=64, beta1=0.5 | loss_mean=0.1344 (±0.0031), accuracy_mean=0.9510 (±0.0014)
2. epochs=40, learning_rate=0.001, batch=32, beta1=0.5 | loss_mean=0.1325 (±0.0034), accuracy_mean=0.9508 (±0.0013)
3. epochs=40, learning_rate=0.001, batch=64, beta1=0.7 | loss_mean=0.1360 (±0.0039), accuracy_mean=0.9505 (±0.0018)
4. epochs=40, learning_rate=0.001, batch=32, beta1=0.7 | loss_mean=0.1350 (±0.0034), accuracy_mean=0.9499 (±0.0015)
5. epochs=40, learning_rate=0.001, batch=32, beta1=0.9 | loss_mean=0.1418 (±0.0024), accuracy_mean=0.9473 (±0.0010)
6. epochs=40, learning_rate=0.001, batch=128, beta1=0.5 | loss_mean=0.1503 (±0.0041), accuracy_mean=0.9461 (±0.0017)
7. epochs=40, learning_rate=0.001, batch=64, beta1=0.9 | loss_mean=0.1460 (±0.0048), accuracy_mean=0.9461 (±0.0018)
8. epochs=40, learning_rate=0.001, batch=128, beta1=0.7 | loss_mean=0.1506 (±0.0024), accuracy_mean=0.9459 (±0.0011)
9. epochs=30, learning_rate=0.001, batch=32, beta1=0.7 | loss_mean=0.1568 (±0.0019), accuracy_mean=0.9418 (±0.0009)
10. epochs=30, learning_rate=0.001, batch=32, beta1=0.5 | loss_mean=0.1578 (±0.0035), accuracy_mean=0.9416 (±0.0017)


Top 10 piores combinações (melhor pro pior):
1. epochs=40, learning_rate=0.1, batch=64, beta1=0.5 | loss_mean=2.1443 (±0.2467), accuracy_mean=0.1303 (±0.0435)
2. epochs=20, learning_rate=0.1, batch=128, beta1=0.5 | loss_mean=2.1389 (±0.2618), accuracy_mean=0.1296 (±0.0448)
3. epochs=20, learning_rate=0.1, batch=64, beta1=0.7 | loss_mean=2.1757 (±0.2527), accuracy_mean=0.1294 (±0.0446)
4. epochs=40, learning_rate=0.1, batch=32, beta1=0.5 | loss_mean=2.1916 (±0.2710), accuracy_mean=0.1243 (±0.0420)
5. epochs=5, learning_rate=0.1, batch=32, beta1=0.7 | loss_mean=2.2089 (±0.2102), accuracy_mean=0.1200 (±0.0386)
6. epochs=20, learning_rate=0.1, batch=32, beta1=0.5 | loss_mean=2.2035 (±0.2245), accuracy_mean=0.1191 (±0.0394)
7. epochs=10, learning_rate=0.1, batch=32, beta1=0.5 | loss_mean=2.2321 (±0.1996), accuracy_mean=0.1145 (±0.0349)
```

```
8. epochs=30, learning_rate=0.1, batch=128, beta1=0.5 | loss_mean=2.2280 (±0.1944), accuracy_mean=0.1142 (±0.0343)
9. epochs=40, learning_rate=0.1, batch=128, beta1=0.5 | loss_mean=2.2280 (±0.1944), accuracy_mean=0.1142 (±0.0343)
```

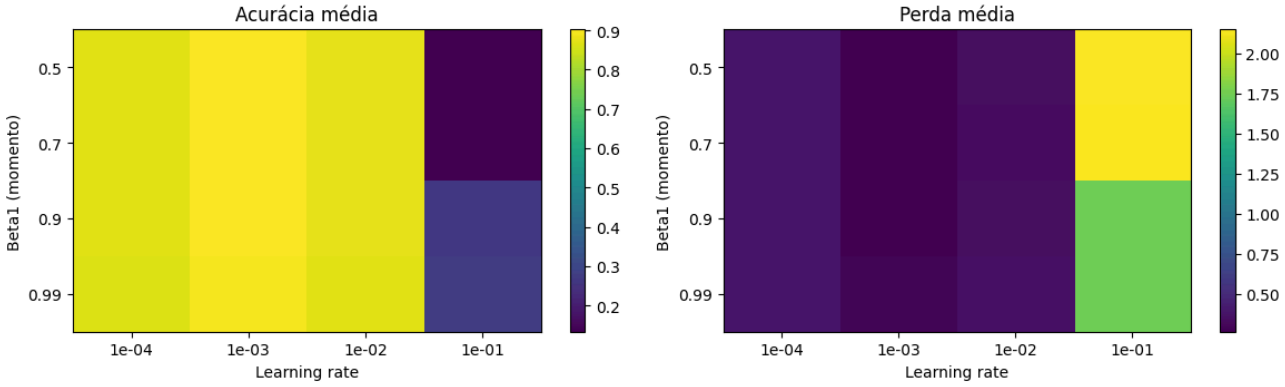
comparações

```
# Loop sobre epochs e batch_size: para cada combinação, gera mapas de calor 2D (beta1 x learning_rate)
# usando as métricas agregadas em `results`.

# Conjuntos ordenados de parâmetros disponíveis em `results`
unique_epochs = sorted(list({r['epochs'] for r in results_q2}))
unique_batch_sizes = sorted(list({r['batch_size'] for r in results_q2}))
unique_beta1s = sorted(list({r['beta1'] for r in results_q2}))
unique_learning_rates = sorted(list({r['learning_rate'] for r in results_q2}))

# Para cada (epochs, batch_size), monta matrizes 2D [beta1 x lr] de acurácia e perda
for epochs in unique_epochs:
    for batch_size in unique_batch_sizes:
        # Filtra resultados referentes à combinação fixa (epochs, batch_size)
        subset = [r for r in results_q2 if r['epochs'] == epochs and r['batch_size'] == batch_size]
        if not subset:
            continue
        # Índices para mapeamento beta1 x lr
        b1_index = {b1: i for i, b1 in enumerate(unique_beta1s)}
        lr_index = {lr: j for j, lr in enumerate(unique_learning_rates)}

        accuracy_matrix = np.full((len(unique_beta1s), len(unique_learning_rates)), np.nan)
        loss_matrix = np.full((len(unique_beta1s), len(unique_learning_rates)), np.nan)

        for r in subset:
            i = b1_index[r['beta1']]
            j = lr_index[r['learning_rate']]
            accuracy_matrix[i, j] = r['accuracy_mean']
            loss_matrix[i, j] = r['loss_mean']

        # Visualização dos mapas de calor
        fig, axes = plt.subplots(1, 2, figsize=(12, 4))
        fig.suptitle(f"Epochs={epochs}, Batch={batch_size}")

        im0 = axes[0].imshow(accuracy_matrix, cmap='viridis', aspect='auto')
        axes[0].set_title('Acurácia média')
        axes[0].set_xticks(range(len(unique_learning_rates)))
        axes[0].set_xticklabels([f"{lr:.0e}" for lr in unique_learning_rates])
        axes[0].set_yticks(range(len(unique_beta1s)))
        axes[0].set_yticklabels([str(b1) for b1 in unique_beta1s])
        axes[0].set_xlabel('Learning rate')
        axes[0].set_ylabel('Beta1 (momento)')
        plt.colorbar(im0, ax=axes[0])

        im1 = axes[1].imshow(loss_matrix, cmap='viridis', aspect='auto')
        axes[1].set_title('Perda média')
        axes[1].set_xticks(range(len(unique_learning_rates)))
        axes[1].set_xticklabels([f"{lr:.0e}" for lr in unique_learning_rates])
        axes[1].set_yticks(range(len(unique_beta1s)))
        axes[1].set_yticklabels([str(b1) for b1 in unique_beta1s])
        axes[1].set_xlabel('Learning rate')
        axes[1].set_ylabel('Beta1 (momento)')
        plt.colorbar(im1, ax=axes[1])

        plt.tight_layout()
        plt.show()
#esperado: loss com cores invertidas de accuracy -> equilibrados
```

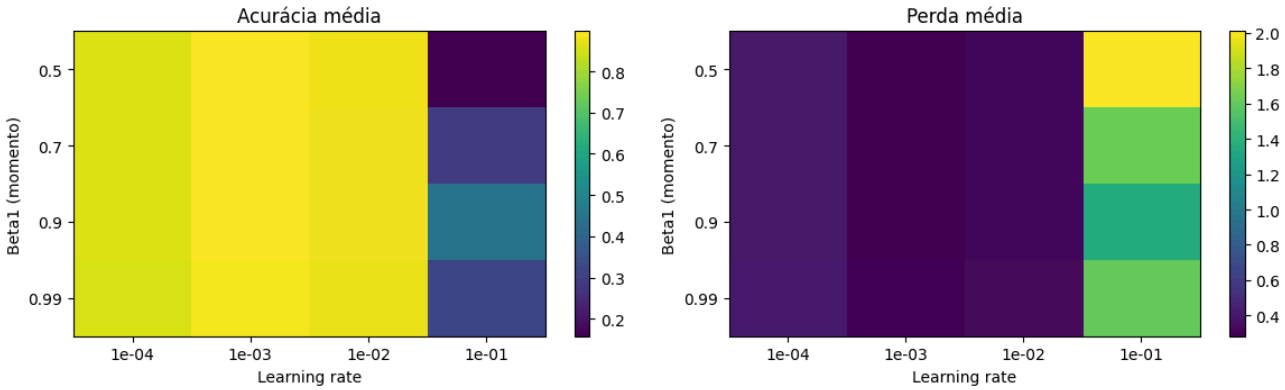

Epochs=5, Batch=32
Acurácia média
Perda média
Beta1 (momento)
Learning rate
1e-04
1e-03
1e-02
1e-01
Epochs=5, Batch=64
Epochs=5, Batch=128
Epochs=5, Batch=256
Epochs=10, Batch=32



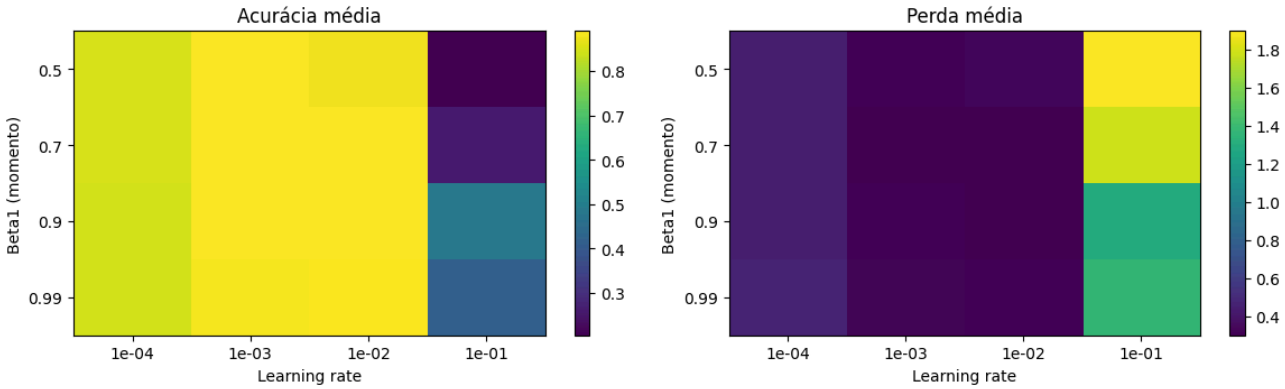
Epochs=10, Batch=64



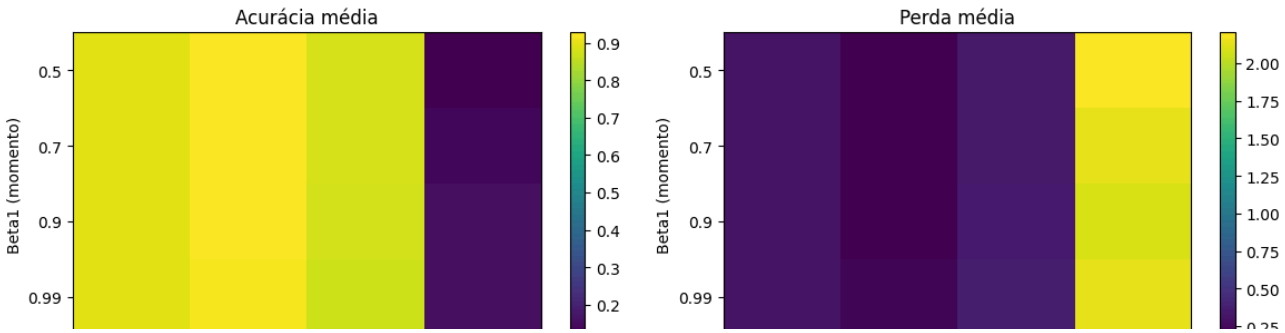
Epochs=10, Batch=128



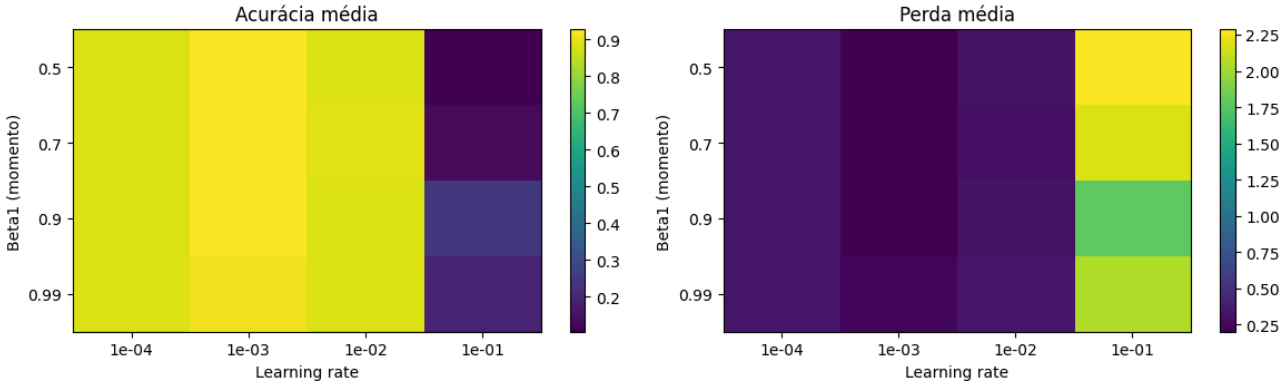
Epochs=10, Batch=256



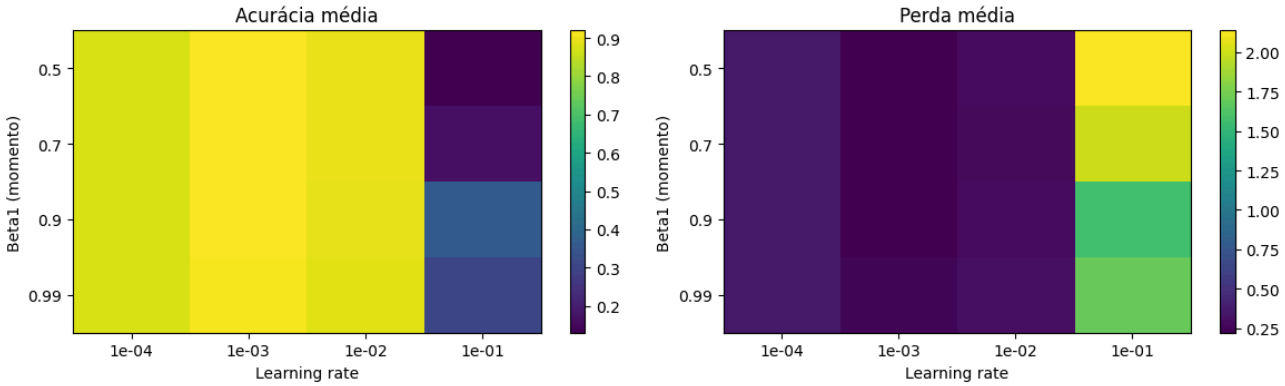
Epochs=20, Batch=32



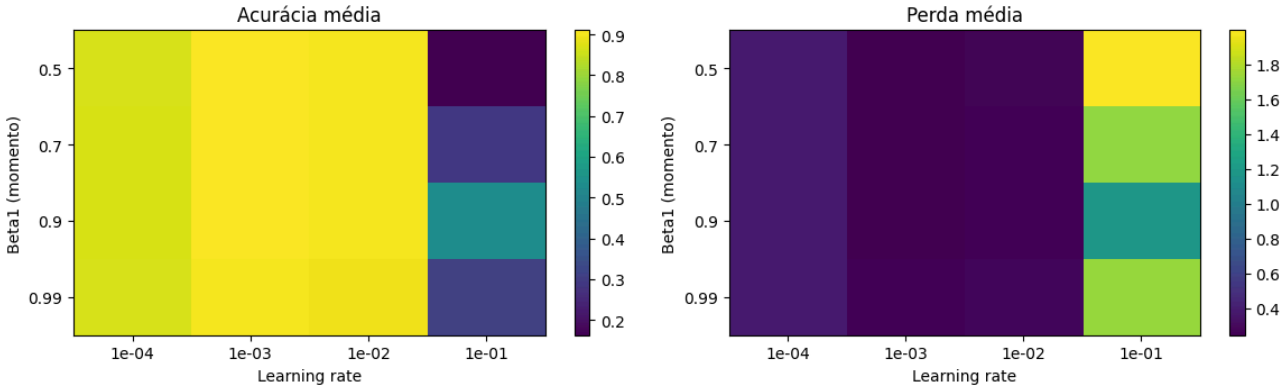
Epochs=20, Batch=64



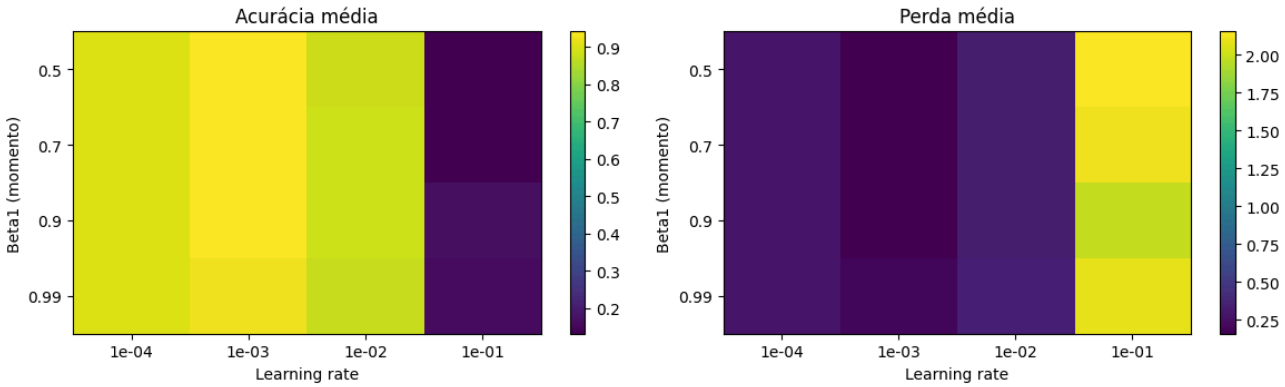
Epochs=20, Batch=128



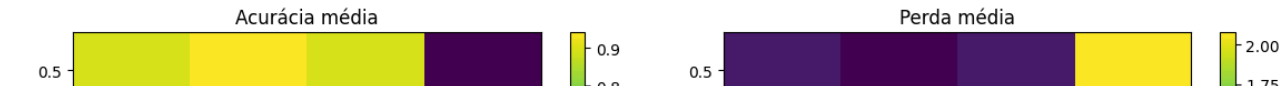
Epochs=20, Batch=256

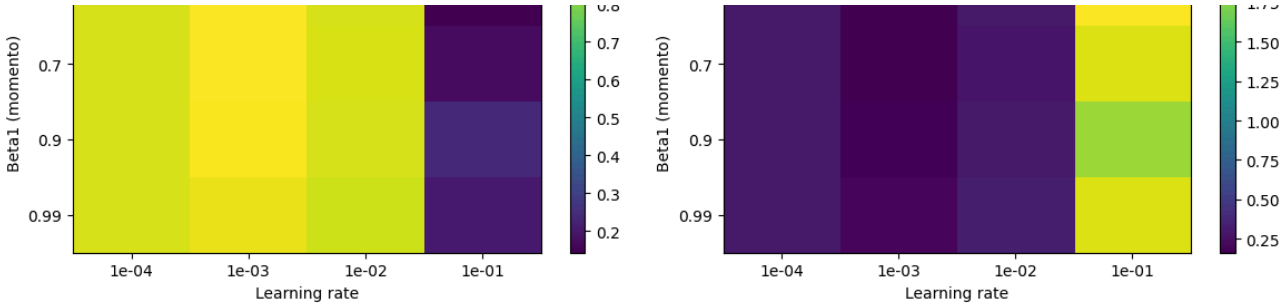


Epochs=30, Batch=32

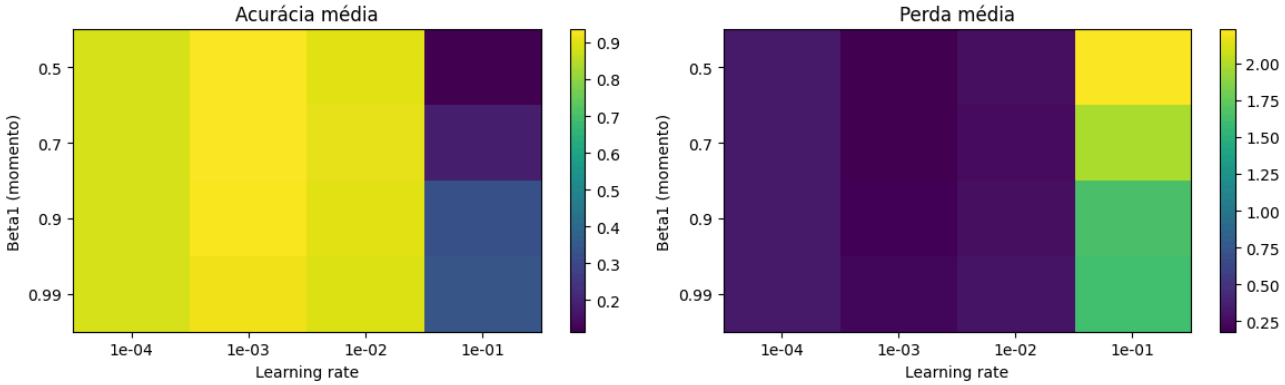


Epochs=30, Batch=64

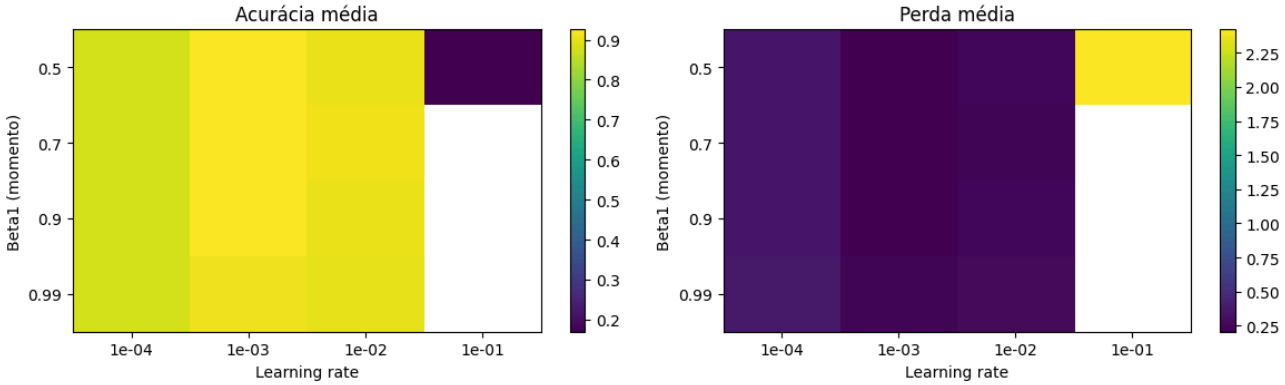




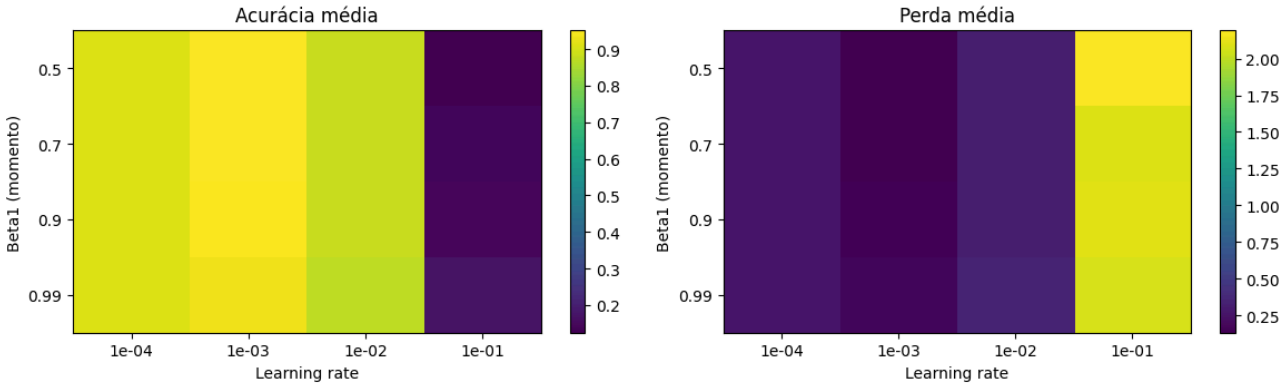
Epochs=30, Batch=128



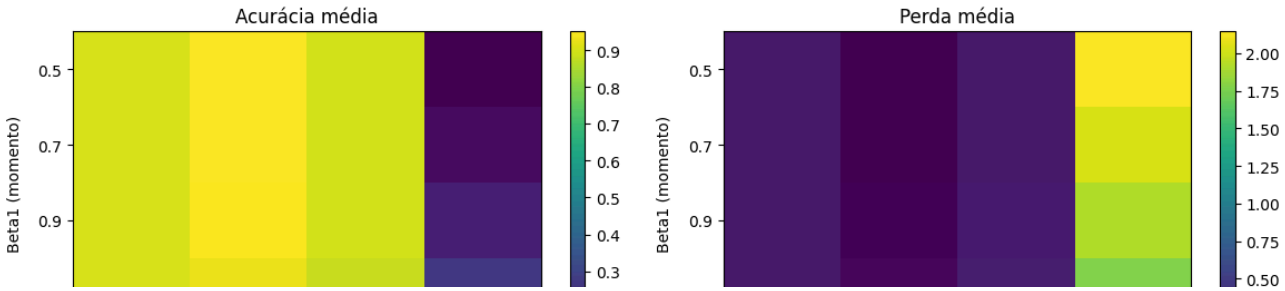
Epochs=30, Batch=256

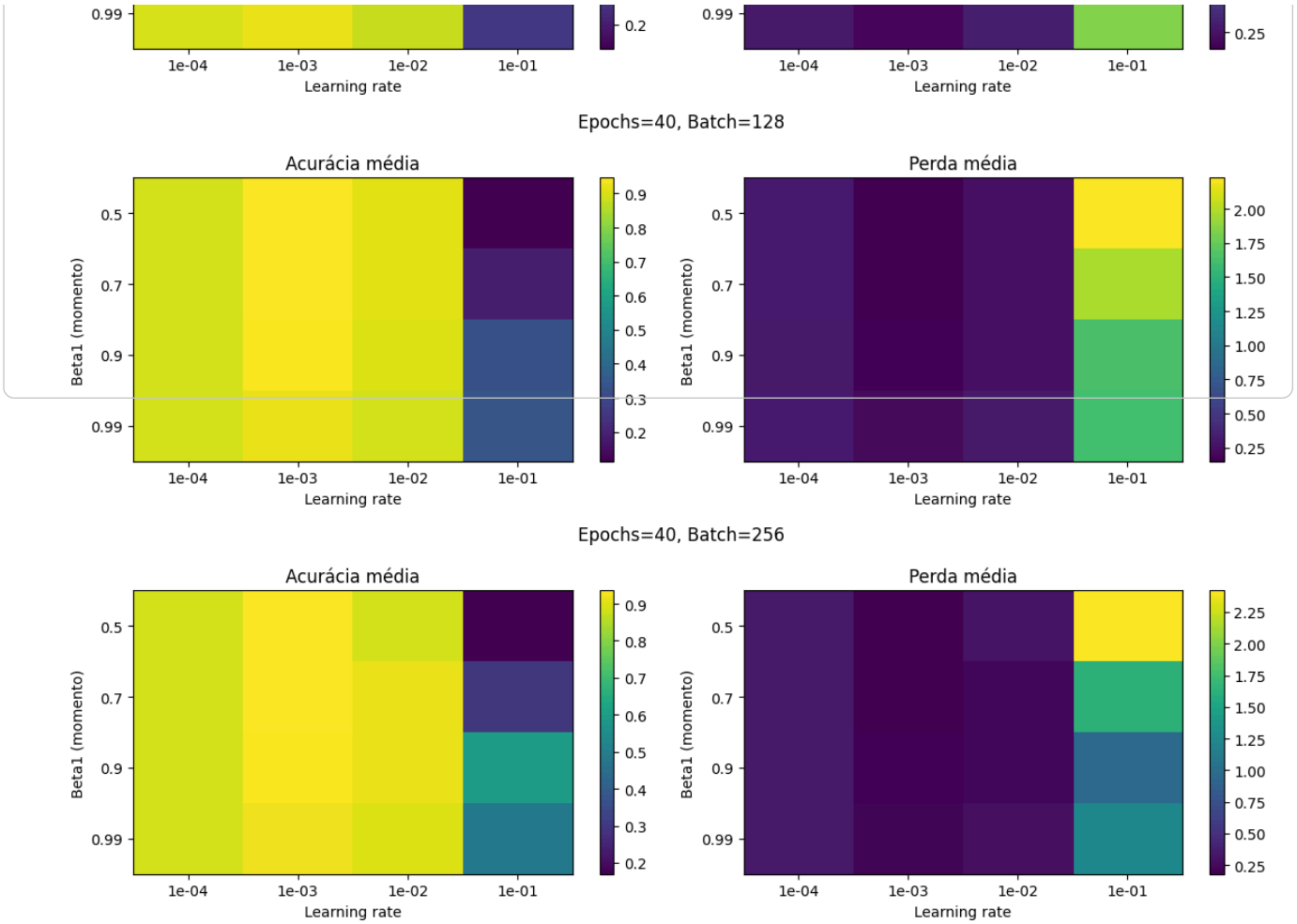


Epochs=40, Batch=32



Epochs=40, Batch=64

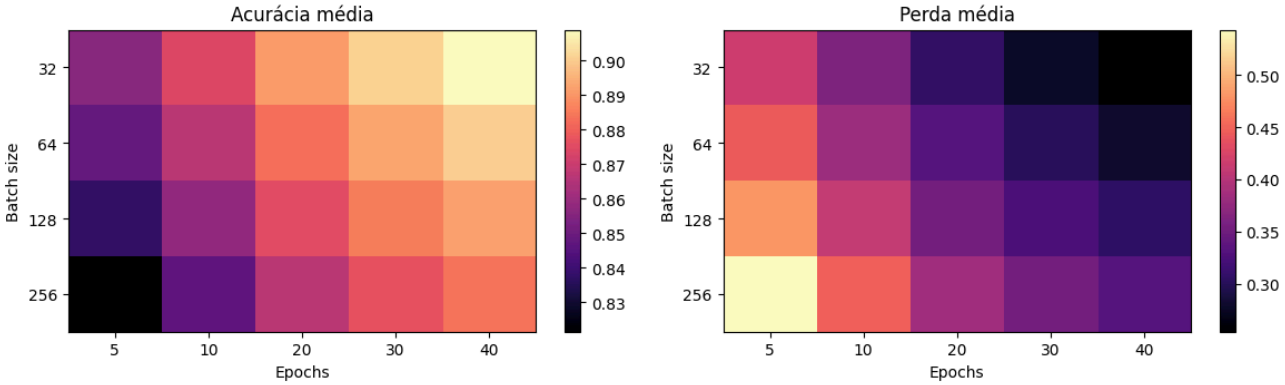



0.99
1e-04
1e-03
1e-02
1e-01
Learning rate
0.2
0.25
Epochs=40, Batch=128
Acurácia média
Perda média
Beta1 (momento)
0.5
0.7
0.9
0.99
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
Epochs=40, Batch=256
2.25

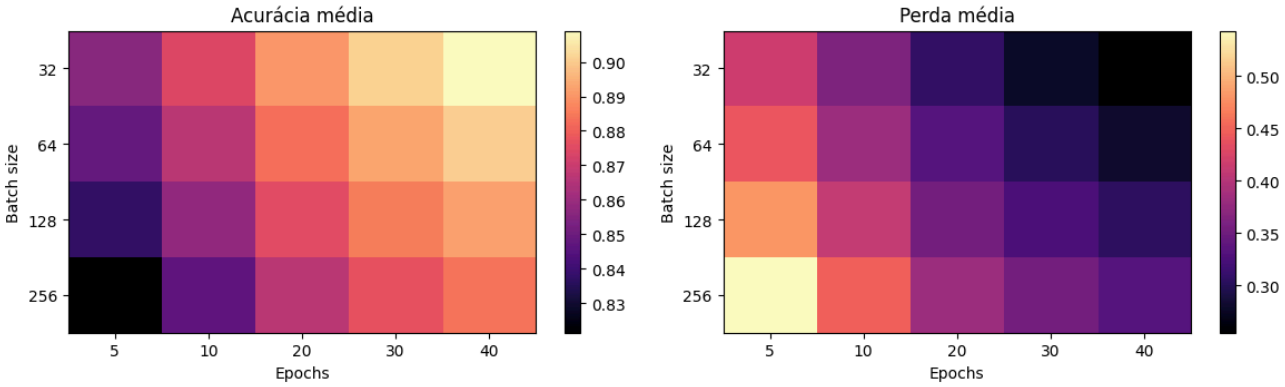
visualização alternativa

```
# Para cada (lr, beta1), monta matrizes 2D [epoch x batch_size] de acurácia e perda
for learning_rate in unique_learning_rates:
    for beta1 in unique_beta1s:
        # Filtra resultados referentes à combinação fixa (epochs, batch_size)
        subset = [r for r in results_q2 if r['learning_rate'] == learning_rate and r['beta1'] == beta1]
        if not subset:
            continue
        # Índices para mapeamento beta1 x lr
        ba_index = {ba: i for i, ba in enumerate(unique_batch_sizes)}
        ep_index = {ep: j for j, ep in enumerate(unique_epochs)}

        accuracy_matrix = np.full((len(unique_batch_sizes), len(unique_epochs)), np.nan)
        loss_matrix = np.full((len(unique_batch_sizes), len(unique_epochs)), np.nan)

        for r in subset:
            i = ba_index[r['batch_size']]
            j = ep_index[r['epochs']]
            accuracy_matrix[i, j] = r['accuracy_mean']
            loss_matrix[i, j] = r['loss_mean']

        # Visualização dos mapas de calor
        fig, axes = plt.subplots(1, 2, figsize=(12, 4))
        fig.suptitle(f"learning_rate={learning_rate}, beta1={beta1}")

        im0 = axes[0].imshow(accuracy_matrix, cmap='magma', aspect='auto')
        axes[0].set_title('Acurácia média')
        axes[0].set_xticks(range(len(unique_epochs)))
        axes[0].set_xticklabels([str(ep) for ep in unique_epochs])
        axes[0].set_yticks(range(len(unique_batch_sizes)))
        axes[0].set_yticklabels([str(b) for b in unique_batch_sizes])
        axes[0].set_xlabel('Epochs')
        axes[0].set_ylabel('Batch size')
        plt.colorbar(im0, ax=axes[0])

        im1 = axes[1].imshow(loss_matrix, cmap='magma', aspect='auto')
        axes[1].set_title('Perda média')
        axes[1].set_xticks(range(len(unique_epochs)))
        axes[1].set_xticklabels([str(ep) for ep in unique_epochs])
        axes[1].set_yticks(range(len(unique_batch_sizes)))
        axes[1].set_yticklabels([str(b) for b in unique_batch_sizes])
        axes[1].set_xlabel('Epochs')
        axes[1].set_ylabel('Batch size')
        plt.colorbar(im1, ax=axes[1])

        plt.tight_layout()
        plt.show()
```

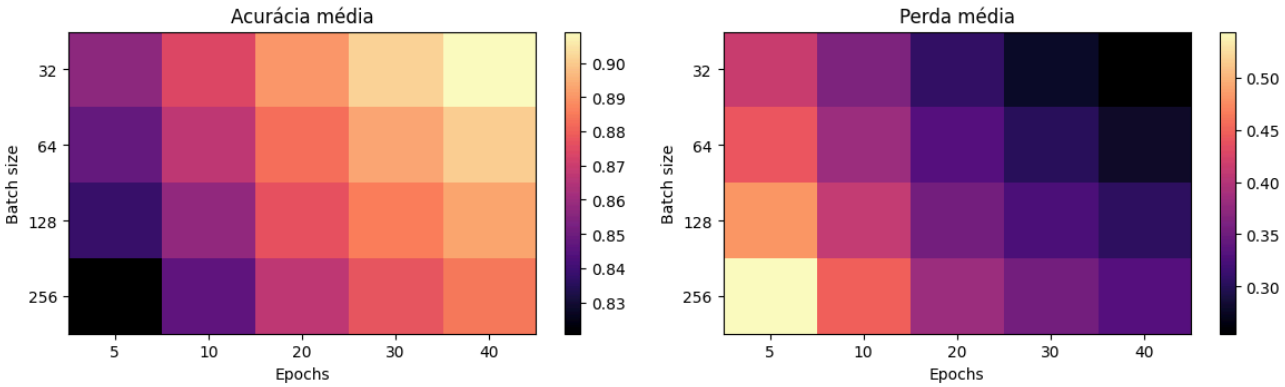
learning_rate=0.0001, beta1=0.5



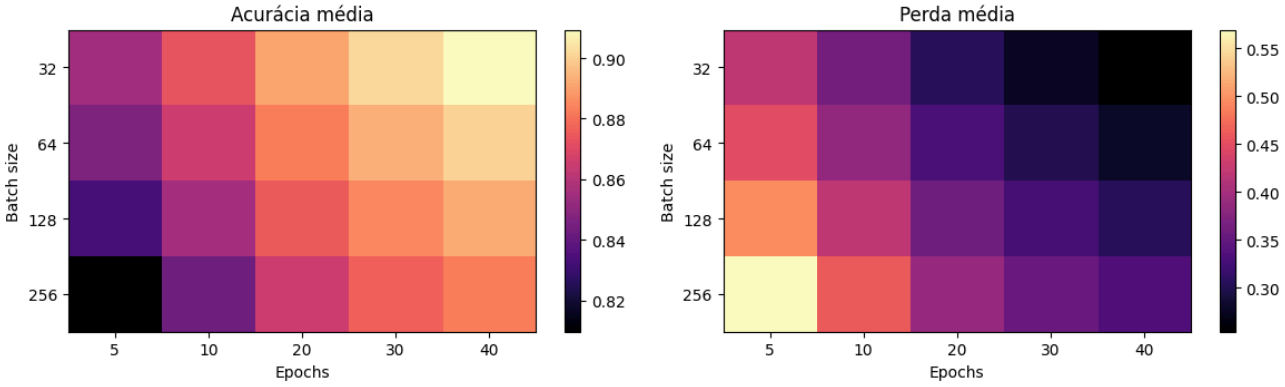
learning_rate=0.0001, beta1=0.7



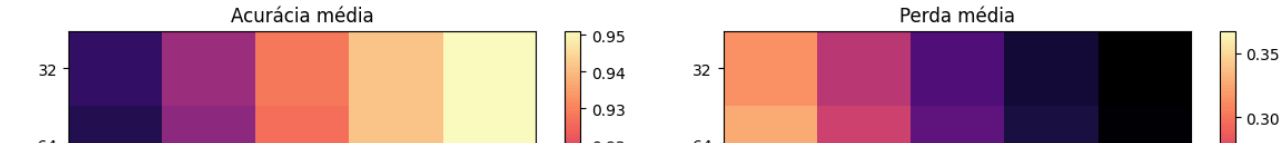
learning_rate=0.0001, beta1=0.9

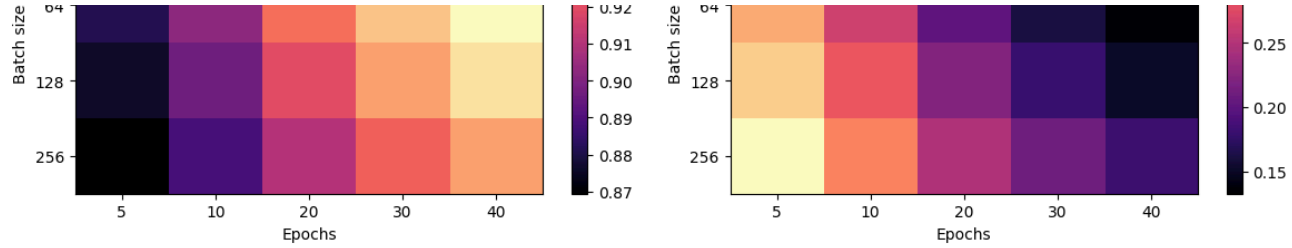


learning_rate=0.0001, beta1=0.99

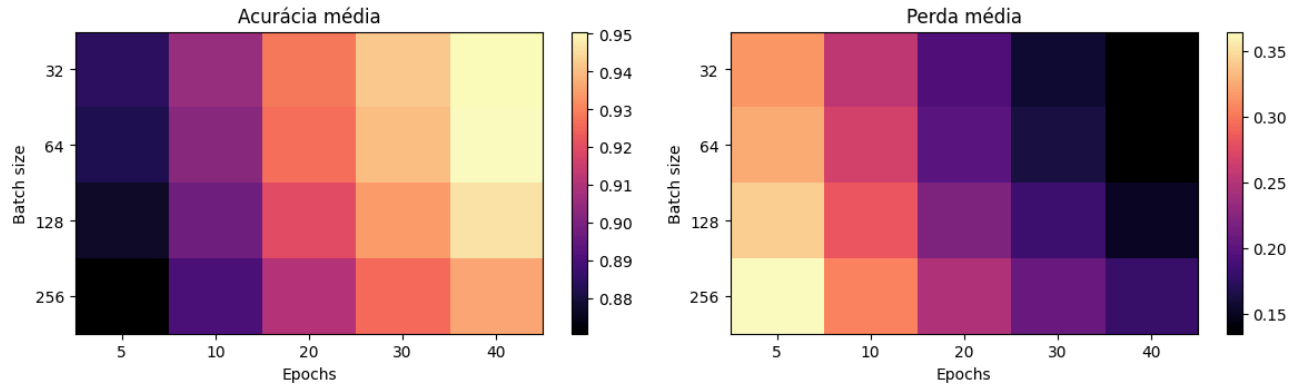


learning_rate=0.001, beta1=0.5

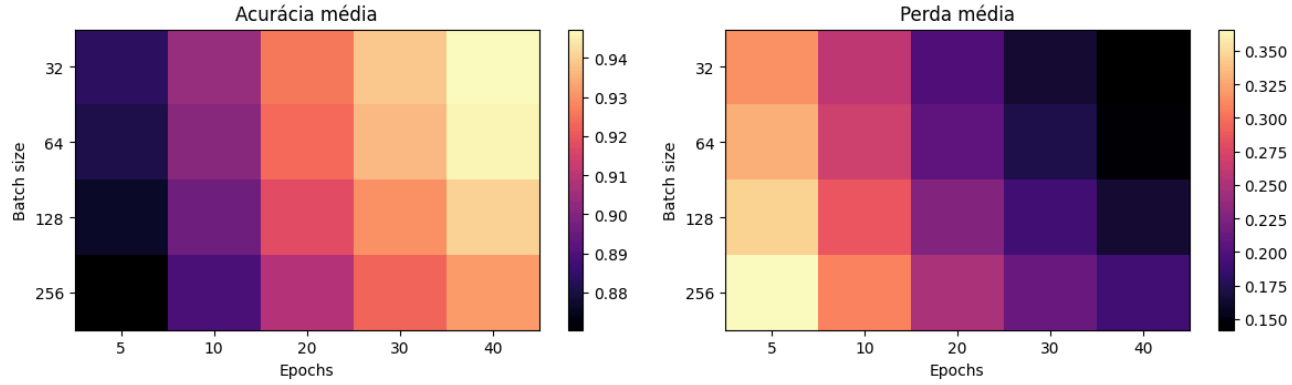




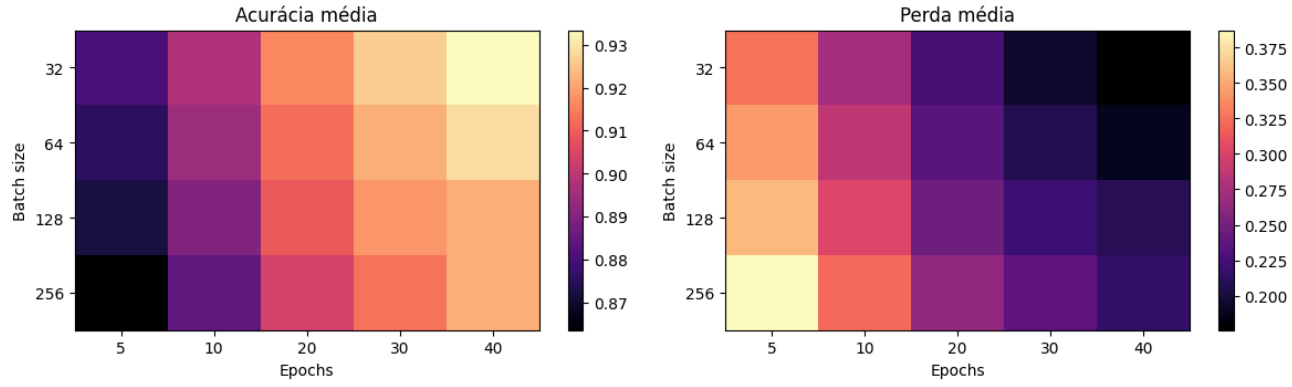
learning_rate=0.001, beta1=0.7



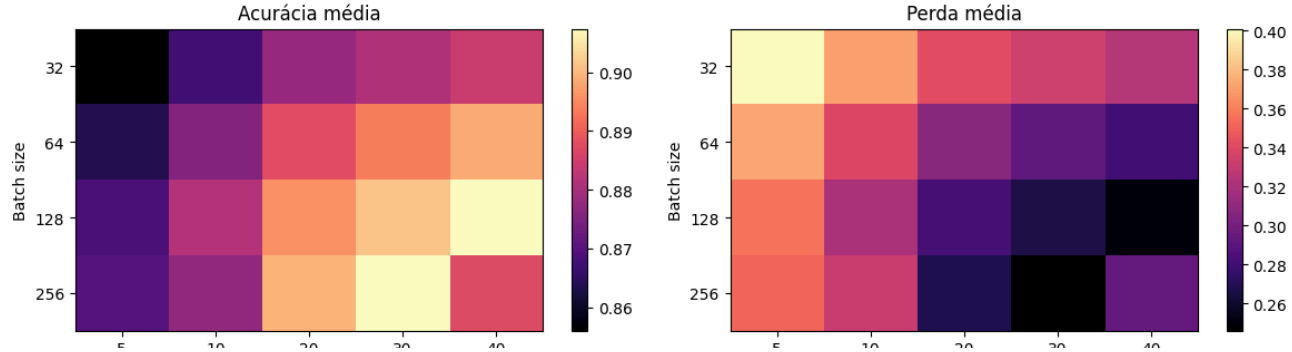
learning_rate=0.001, beta1=0.9



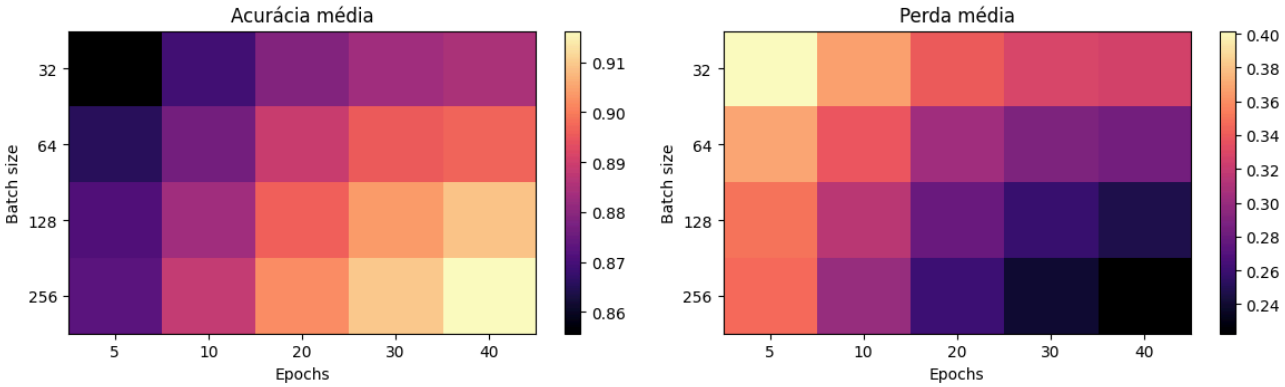
learning_rate=0.001, beta1=0.99



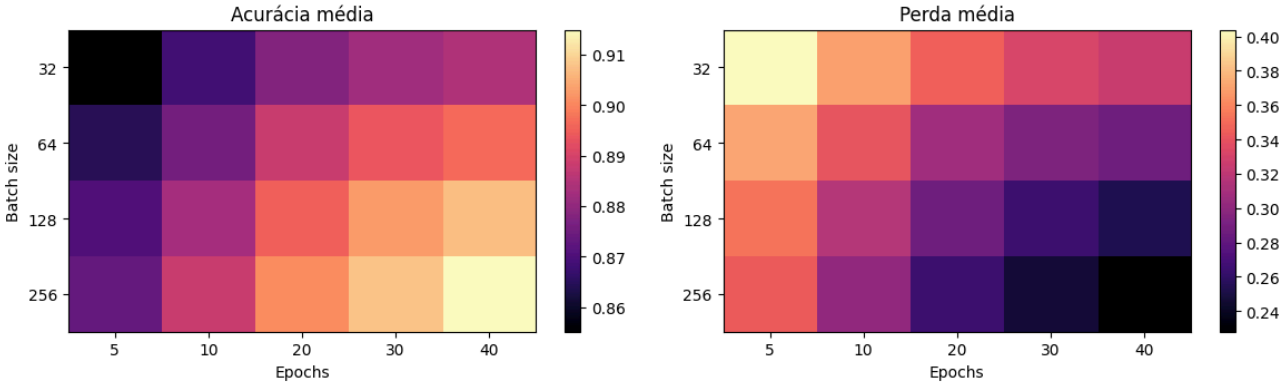
learning_rate=0.01, beta1=0.5



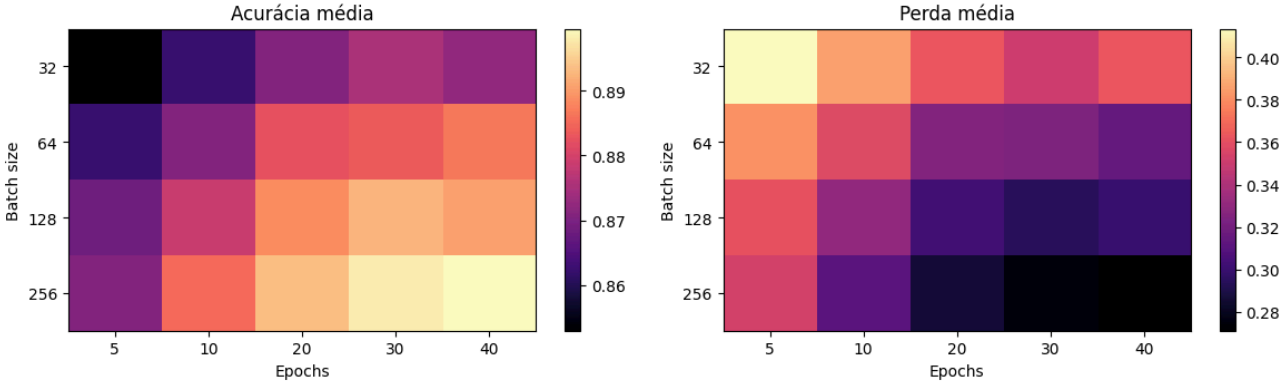
learning_rate=0.01, beta1=0.7



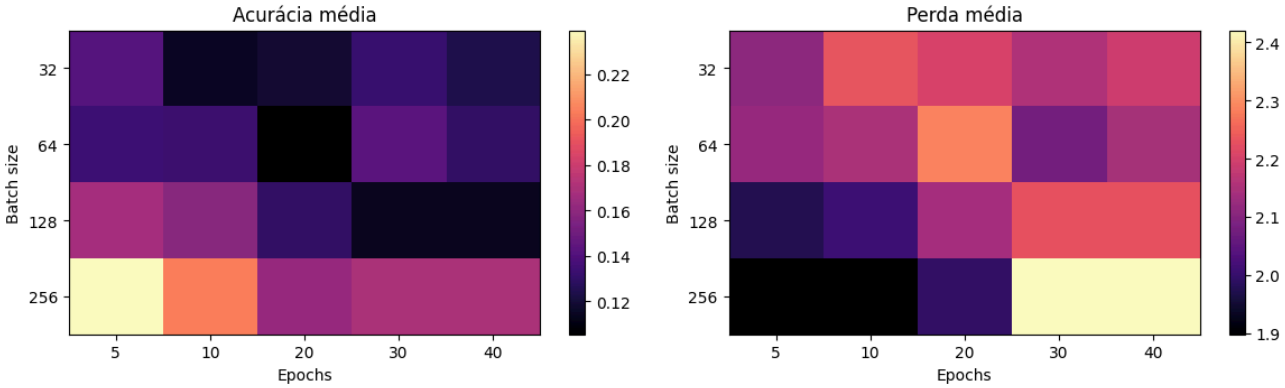
learning_rate=0.01, beta1=0.9



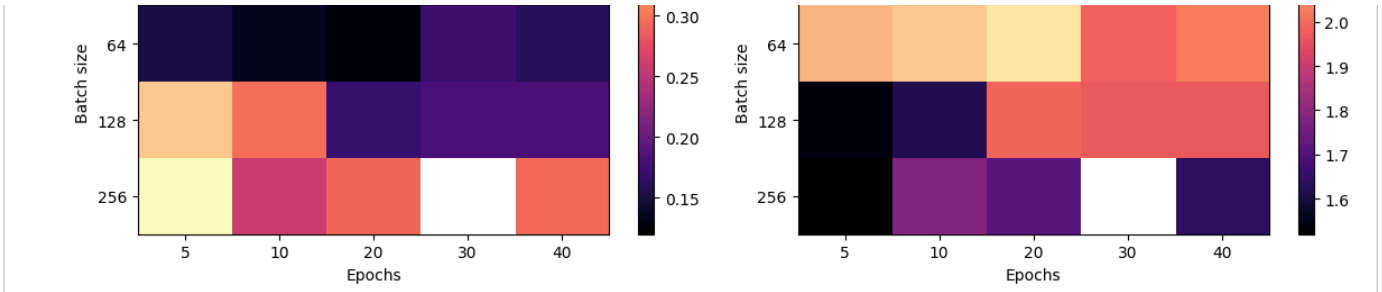
learning_rate=0.01, beta1=0.99



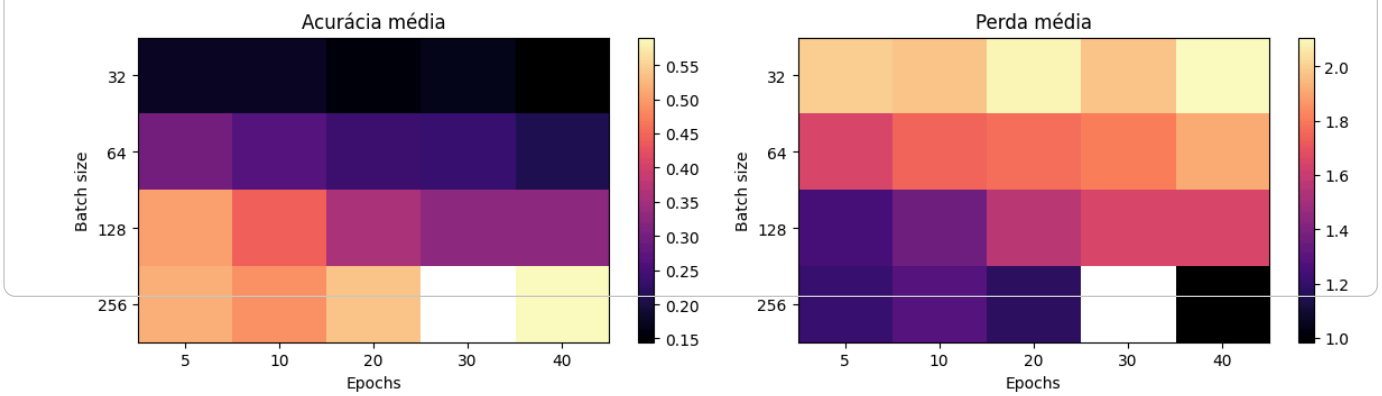
learning_rate=0.1, beta1=0.5



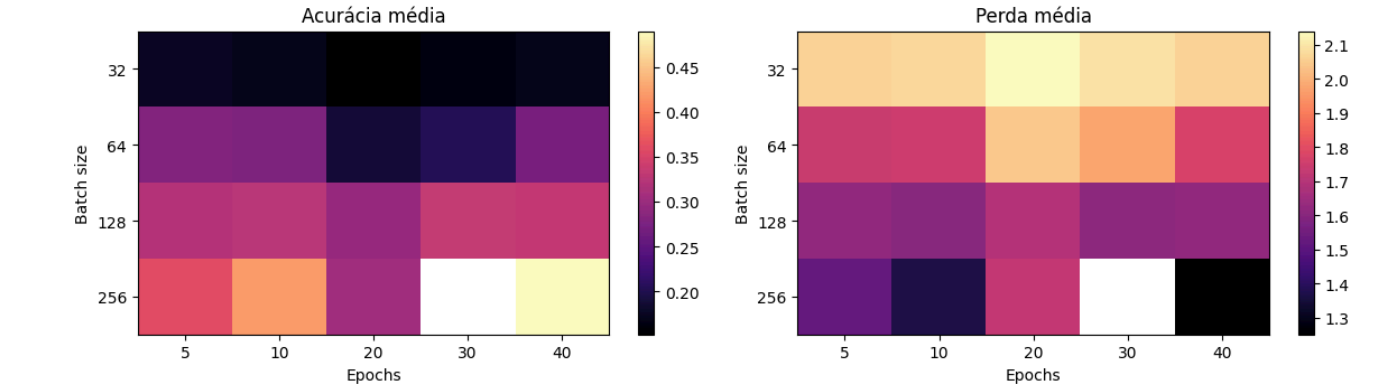
learning_rate=0.1, beta1=0.7



learning_rate=0.1, beta1=0.9



learning_rate=0.1, beta1=0.99

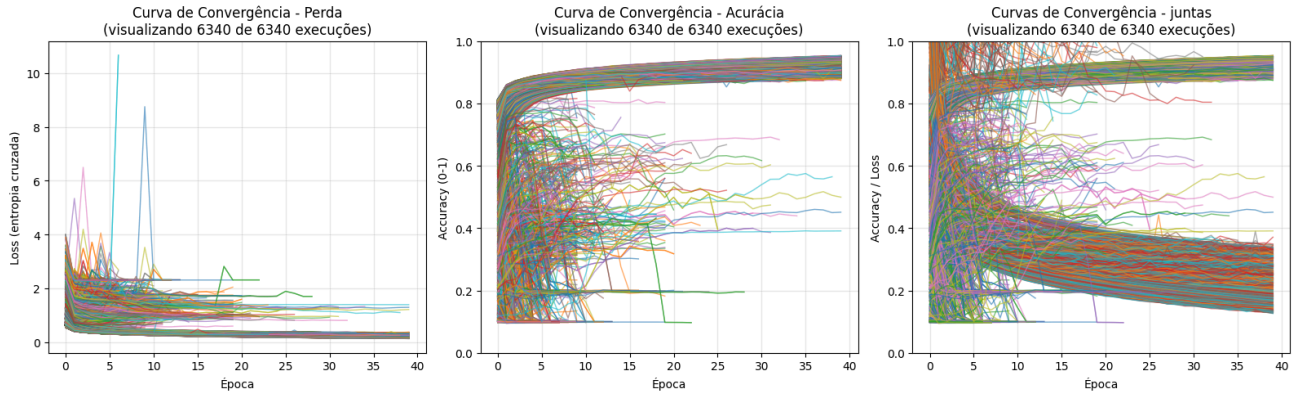


métricas

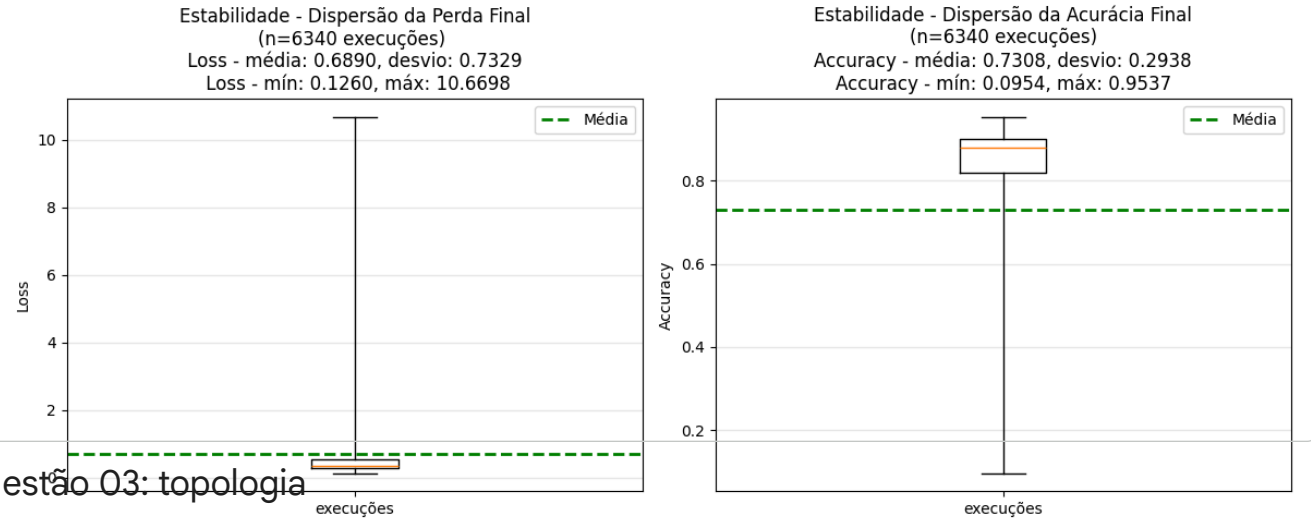
```
print(f"Total de combinações testadas: {len(results_q2)}")


print("========== CURVAS DE CONVERGÊNCIA ==========")
sample_step = 1 # mostra modelos 1 a 1, ajuste para visualização menos poluída
sample_indices = list(range(0, len(histories_q2), sample_step)) #start, stop, step

fig, axes = plt.subplots(1, 3, figsize=(16, 5))

#perda
for idx in sample_indices:
    h = histories_q2[idx]
    axes[0].plot(h.history['loss'], alpha=0.6, linewidth=1)
axes[0].set_title(f'Curva de Convergência - Perda\n(visualizando {len(sample_indices)} de {len(histories_q2)} execuç
axes[0].set_xlabel('Época')
axes[0].set_ylabel('Loss (entropia cruzada)')
axes[0].grid(True, alpha=0.3)

#acurácia
for idx in sample_indices:
    h = histories_q2[idx]
    axes[1].plot(h.history['accuracy'], alpha=0.6, linewidth=1)
axes[1].set_title(f'Curva de Convergência - Acurácia\n(visualizando {len(sample_indices)} de {len(histories_q2)} exe
axes[1].set_xlabel('Época')
axes[1].set_ylabel('Accuracy (0-1)')
axes[1].grid(True, alpha=0.3)
axes[1].set_ylim([0, 1])

#as duas
for idx in sample_indices:
    h = histories_q2[idx]
    axes[2].plot(h.history['accuracy'], alpha=0.6, linewidth=1)
    axes[2].plot(h.history['loss'], alpha=0.6, linewidth=1)
axes[2].set_title(f'Curvas de Convergência - juntas\n(visualizando {len(sample_indices)} de {len(histories_q2)} exec
axes[2].set_xlabel('Época')
axes[2].set_ylabel('Accuracy / Loss')
axes[2].grid(True, alpha=0.3)
axes[2].set_ylim([0, 1])

plt.tight_layout()
plt.show()


train_losses = [h.history['loss'][-1] for h in histories_q2]
train_accuracies = [h.history['accuracy'][-1] for h in histories_q2]
print(f"\n========== ESTABILIDADE (n={len(train_losses)}) ==========")


fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Boxplot de Loss
axes[0].boxplot(train_losses, whis=(0, 100))
axes[0].set_title(f'Estabilidade - Dispersão da Perda Final\n(n={len(train_losses)} execuções) '
                  f'\n Loss - média: {np.mean(train_losses):.4f}, desvio: {np.std(train_losses):.4f} '
                  f'\n Loss - mín: {np.min(train_losses):.4f}, máx: {np.max(train_losses):.4f}')
axes[0].set_ylabel('Loss')
axes[0].set_xticklabels(['execuções'])
axes[0].axhline(y=np.mean(train_losses), color='green', linestyle='--', linewidth=2, label='Média')
#pontos individuais
#axes[0].scatter([1]*len(train_losses), train_losses, color='red', zorder=2)
axes[0].legend()
axes[0].grid(True, alpha=0.3, axis='y')

# Boxplot de Accuracy
axes[1].boxplot(train_accuracies, whis=(0, 100))
axes[1].set_title(f'Estabilidade - Dispersão da Acurácia Final\n(n={len(train_accuracies)} execuções)'
                  f'\nAccuracy - média: {np.mean(train_accuracies):.4f}, desvio: {np.std(train_accuracies):.4f}'
                  f'\nAccuracy - mín: {np.min(train_accuracies):.4f}, máx: {np.max(train_accuracies):.4f}')
axes[1].set_ylabel('Accuracy')
axes[1].set_xticklabels(['execuções'])
axes[1].axhline(y=np.mean(train_accuracies), color='green', linestyle='--', linewidth=2, label='Média')
#pontos individuais
#axes[1].scatter([1]*len(train_accuracies), train_accuracies, color='red', zorder=2)
```

```
axes[1].legend()
axes[1].grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()


print("\n========== TEMPO DE TREINAMENTO ==========")

all_times = [r['time_mean'] for r in results_q2]
all_time_stds = [r['time_std'] for r in results_q2]
#média e desvio do tempo de execução do mesmo modelo para todas as seeds

print(f"Tempo médio geral: {np.mean(all_times):.2f}s (±{np.std(all_times):.2f}s)")
print(f"Tempo mínimo: {np.min(all_times):.2f}s")
print(f"Tempo máximo: {np.max(all_times):.2f}s")

#tempo por quantidade total de épocas do modelo
time_by_epochs = {}
for r in results_q2:
    ep = r['epochs']
    if ep not in time_by_epochs:
        time_by_epochs[ep] = []
    time_by_epochs[ep].append(r['time_mean'])

print("\nTempo médio por número de épocas:")
for ep in sorted(time_by_epochs.keys()):
    print(f"  {ep} épocas: {np.mean(time_by_epochs[ep]):.2f}s (±{np.std(time_by_epochs[ep]):.2f}s)")

# Gráfico de tempo por épocas
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

epochs_list = sorted(time_by_epochs.keys())
mean_times = [np.mean(time_by_epochs[ep]) for ep in epochs_list]
std_times = [np.std(time_by_epochs[ep]) for ep in epochs_list]

axes[0].set_title('Tempo de Treinamento vs Número de Épocas')
axes[0].bar(epochs_list, mean_times, yerr=std_times, alpha=0.7, capsize=10, color='teal')
axes[0].set_xlabel('Número de Épocas')
axes[0].set_ylabel('Tempo Médio de Treinamento (s)')
axes[0].grid(True, alpha=0.3, axis='y')


axes[1].set_title('Distribuição dos Tempos de Treinamento')
axes[1].hist(all_times, bins=25, alpha=0.7, color='teal', edgecolor='black')
axes[1].axvline(np.mean(all_times), color='red', linestyle='--', linewidth=2, label=f'Média: {np.mean(all_times):.2f
axes[1].set_xlabel('Tempo de Treinamento (s)')
axes[1].set_ylabel('Frequência')
axes[1].legend()
axes[1].grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()


print("\n========== TOP 5 COMBINAÇÕES MAIS RÁPIDAS ==========")
sorted_by_time = sorted(results_q2, key=lambda x: x['time_mean'])
for i, r in enumerate(sorted_by_time[:5], 1):
    print(f"{i}. Tempo: {r['time_mean']:.2f}s | epochs={r['epochs']}, lr={r['learning_rate']}, "
          f"batch={r['batch_size']}, beta1={r['beta1']}")
    print(f"   Loss: {r['loss_mean']:.4f}, Acc: {r['accuracy_mean']:.4f}")

print("\n========== TOP 5 COMBINAÇÕES MAIS LENTAS ==========")
for i, r in enumerate(sorted_by_time[-5:], 1):
    print(f"{i}. Tempo: {r['time_mean']:.2f}s | epochs={r['epochs']}, lr={r['learning_rate']}, "
          f"batch={r['batch_size']}, beta1={r['beta1']}")
    print(f"   Loss: {r['loss_mean']:.4f}, Acc: {r['accuracy_mean']:.4f}")
```

```
Total de combinações testadas: 317
========== CURVAS DE CONVERGÊNCIA ==========
```




```
========== ESTABILIDADE (n=6340) ==========
```



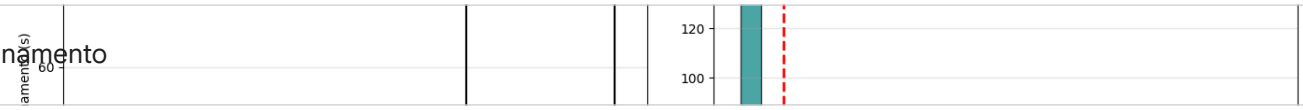
Questão 03: topologia

melhor combinação: activation function = sigmoid epochs=40 learning_rate=0.001 batch=64 beta1=0.5

```
========== TEMPO DE TREINAMENTO ==========
Tempo médio geral: 24.26s (±29.71s)
Tempo mínimo: 2.28s
Tempo máximo: 270.15s

Tempo médio por número de épocas:
```

Parâmetros ajustados

```
num_hidden_layers_options = [1, 2, 3, 4]
neurons_per_layer_options = {
    1: [[32], [64], [128], [256]],
    2: [[32, 16], [64, 32], [128, 64], [256, 128], [64, 64], [32, 64]],
    3: [[128, 64, 32], [256, 128, 64], [512, 256, 128], [64, 64, 64], [32, 64, 128]],
    4: [[256, 128, 64, 32], [512, 256, 128, 64], [1024, 512, 256, 128], [64, 64, 64, 64], [32, 64, 128, 256]]
}
```



treinamento

```
#TODO: aumentar número de seeds para teste exaustivo final
#TODO: treino e validação
import time
from sklearn.metrics import f1_score, precision_score, recall_score

seeds_q3 = spaced_seeds(20, base, PRIME_STEP)

#checkpoint configs
checkpoint_dir = Path('checkpoints')
checkpoint_file = checkpoint_dir / 'results_q3_checkpoint_1.pkl'
results_q3, histories_q3, start_combo = load_checkpoint(checkpoint_file, 'q3')
current_combination = 0
total_combinations = sum(len(neurons_per_layer_options[n]) for n in num_hidden_layers_options)
print(f"Total de combinações a testar: {total_combinations}")

for num_hidden_layers in num_hidden_layers_options:
    for neurons_per_layer in neurons_per_layer_options[num_hidden_layers]:
```