Otimização Avançada, Diagnóstico de Gradientes e TensorBoard

Aula 08 — Regressão em Imagens (UTKFace/Kaggle), AdamW, Schedulers e Observabilidade

Instrutor: **Prof. Allan Spadini**

Redes Neurais Profundas | Graduação Faculdade Infnet

Caso Prático (Kaggle): Estimativa de Idade Facial (UTKFace)

Regressão contínua em imagens RGB já recortadas — pré-processamento simples para focar no treinamento

KAGGLE • VISÃO COMPUTACIONAL

Dataset UTKFace (Kaggle): Estimativa Contínua de Idade por Imagem Facial

23.708 ROSTOS RGB • IDADES 1–116 ANOS

1. Rostos Alinhados (RGB)



Infância

`6_1_0_*.jpg → 6 anos`



Vida Adulta

`32_0_2_*.jpg → 32 anos`



Terceira Idade

`74_1_0_*.jpg → 74 anos`

2. AgeCNN (PyTorch)

`Tensor (N, 3, 64, 64)`

↓

3× (Conv2d + BatchNorm2d + ReLU + Pool)
`+ AdaptiveAvgPool2d(4×4)`

↓

Cabeça Regressora (Sem Softmax)
`Linear(128, 1) → Idade (Anos)`

3. Aplicações & Foco da Aula

KYC, Verificação Etária & Varejo
Estima idade contínua direto da foto facial

Alvo Contínuo: idade (anos)
`min=1 | média=33.3 | max=116 anos`

Foco Principal: TensorBoard
Auditoria visual de treino, grafo e runs!

PIPELINE LIMPO DO KAGGLE (SEM PRÉ-PROCESSAMENTO COMPLEXO):

1. `kagglehub.dataset_download`
Download em 1 linha sem API key

2. `nome.split("_")[0]`
Idade extraída direto do nome .jpg

3. `torchvision.transforms`
Mesma receita da Aula 07 (64×64)

4. `Split 70% / 15% / 15%`
Pessoas independentes (i.i.d.)

Rostos já recortados (`200×200` RGB) e rótulo no nome do arquivo eliminam burocracia e liberam a aula para o TensorBoard.

Da Classificação (Aula 07) à Regressão Visual (AgeCNN)

Reaproveitando o pipeline de imagens da Aula 07 e trocando apenas a cabeça de saída

ARQUITETURA & FUNÇÃO DE PERDA

Cabeça de Classificação (Aula 07) vs. Cabeça de Regressão (Aula 08)

MESMO PIPELINE TORCHVISION.TRANSFORMS

AULA 07: CLASSIFICAÇÃO (EUROSAT)

Backbone CNN	→	`nn.Linear(256, 10)`	→	`CrossEntropyLoss()`	→	Métricas
`Flatten → 256 feats`		10 logits (1 por classe)		Softmax interno → Probs		Acurácia / F1

AULA 08: REGRESSÃO VISUAL (UTKFACE • KAGGLE)

CNN + AdaptivePool(4×4)	→	`nn.Linear(128, 1)`	→	`nn.MSELoss()`	→	Negócio (Anos)
`64×4×4 = 1.024 → 128`		**Sem ReLU/Sigmoid na saída!**		Erro quadrático $(\hat{y} - y)^2$		**MAE, RMSE, R^2**

✅ Pré-processamento Reutilizado (Receita da Aula 07)

Mesmo pipeline padrão com torchvision.transforms sem dor de cabeça:

`Resize(64×64) → RandomHorizontalFlip → ToTensor → Normalize`

🎯 Cabeça Regressora & Auditoria Visual

Saída linear contínua prevê a idade em anos sem cortar extremos.

`MAE em anos + logs completos por experimento no TensorBoard!`

Reaproveitamos exatamente o `transforms` da Aula 07 e trocamos apenas a cabeça final para prever idade contínua.

A Física dos Otimizadores: SGD, Momentum e AdamW

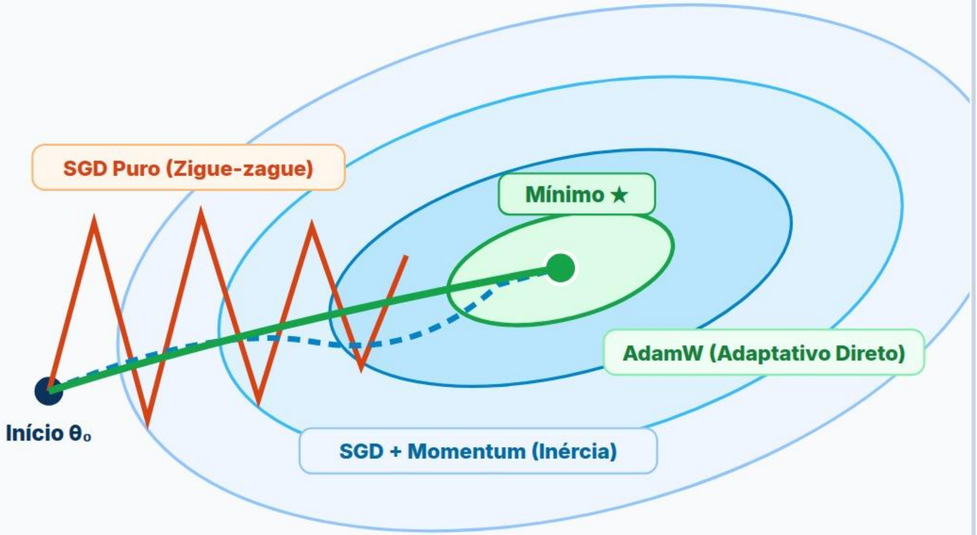
Como inércia e taxas adaptativas vencem vales estreitos na superfície de perda

DINÂMICA DE OTIMIZAÇÃO

Trajetórias na Superfície de Perda: SGD vs. Momentum vs. AdamW

ADAMW = PADRÃO MODERNO

Curvas de Nível da Loss (Vale Estreito / Ravina)



Sem curvatura compensada, o SGD oscila nas paredes; AdamW escala cada peso individualmente.

1. SGD Puro

$w \leftarrow w - \eta \cdot \nabla L$

- Mesma taxa η para todos os parâmetros.
- Lento em vales estreitos (alta oscilação).

2. SGD + Momentum (β = 0.9)

$v \leftarrow \beta \cdot v + \nabla L \mid w \leftarrow w - \eta \cdot v$

- Soma velocidade passada (inércia física).
- Cancela zigue-zague lateral e acelera no vale.

3. AdamW (Momentos 1ª e 2ª Ordem)

$w \leftarrow w - \eta \cdot (\hat{m} / \sqrt{\hat{v}}) - \eta \cdot \lambda \cdot w$

- Passo individualizado por variância ($\hat{v}$).
- **Weight Decay (λ·w) desacoplado do gradiente!**

O AdamW corrige o Adam original aplicando a regularização L2 diretamente sobre os pesos, tornando-o o padrão em CNNs modernas.

Simulador Interativo: Agendadores de Taxa de Aprendizado

Compare como diferentes políticas de LR afetam a velocidade e o refinamento da perda

Agendadores de LR (Schedulers)

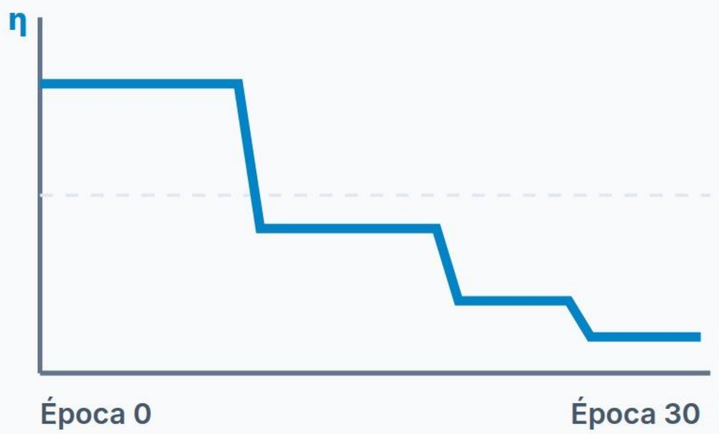
Estratégia de Decaimento:

- **ReduceLROnPlateau** (selecionado)
- Cosine + Warmup
- OneCycleLR
- StepLR (Degraus)
- LR Constante

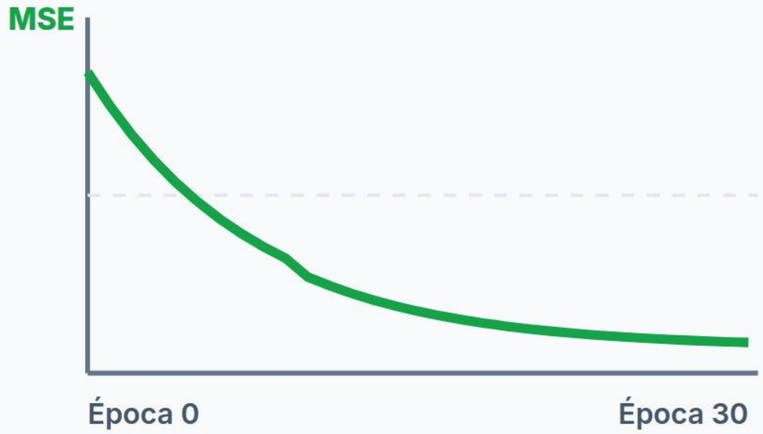
Taxa Inicial (η_0): `1.5e-3`

Comportamento: ReduceLROnPlateau monitora val_loss: se estagnar por patience épocas, reduz η pela metade para refinar a regressão.

Trajetória da Taxa de Aprendizado η(t)



Convergência do Erro (Validação MSE)



Regra prática: explore rápido com η_0 no início e reduza a velocidade com `ReduceLROnPlateau` para pousar no mínimo sem oscilar.

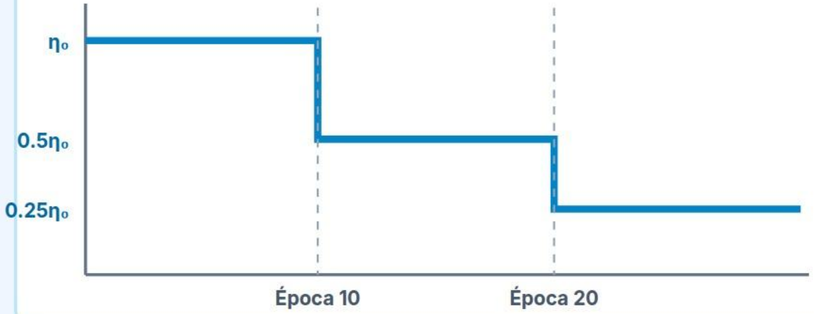
LR SCHEDULING NO PYTORCH

StepLR (Programado por Relógio) vs. ReduceLROnPlateau (Reativo à Validação)

REFINAMENTO NO MÍNIMO

STEPLR (DECAIMENTO FIXO)

Reduz no relógio a cada step_size épocas

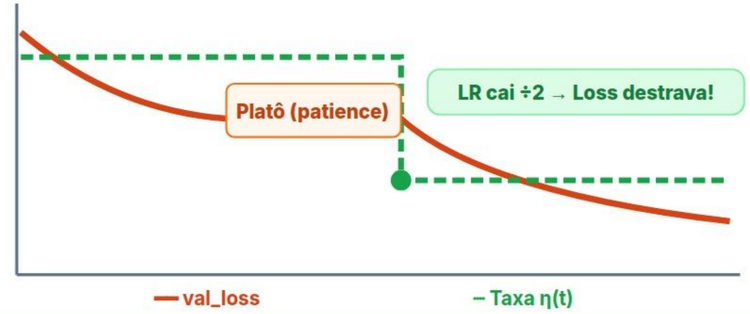



```
StepLR(optimizer, step_size=10, gamma=0.5)
```

- Chamado com scheduler.step() após cada época.
- Simples e previsível, mas cego à perda real.

REDUCELRONPLATEAU (FEEDBACK)

Reage somente quando val_loss estagna




```
ReduceLROnPlateau(opt, mode="min", patience=2)
```

- Chamado com scheduler.step(val_loss).
- **Ideal para regressão real: só desacelera quando precisa!**

No notebook da Aula 08, o `ReduceLROnPlateau` reduz o passo automaticamente quando o MAE de validação estaciona.

Termômetro da Norma dos Gradientes ($||\nabla L||_2$)

Medindo em um único número a intensidade das atualizações durante a retropropagação

DIAGNÓSTICO NUMÉRICO

Termômetro da Norma Euclidiana dos Gradientes

MONITORAMENTO POR ÉPOCA

$$\|\nabla_\theta L\|_2 = \sqrt{\sum_{p \in \theta} \sum_i \left(\frac{\partial L}{\partial p_i}\right)^2}$$

O que mede? Comprimento total do passo combinado de todas as camadas.
Em PyTorch: `sum(p.grad.norm(2).item()**2)**0.5`

Faixas de Diagnóstico do Termômetro $||\nabla L||_2$

VANISHING ($< 10^{-4}$)	FAIXA SAUDÁVEL (0.1 a 3.0)	EXPLODING (> 10.0)
Sintoma: Rede Congelada	**Sintoma: Convergência Estável**	**Sintoma: Spikes e Risco de NaN**
• Derivadas somem rumo à Conv1. • Loss estaciona logo no início.	• Todas as camadas recebem sinal útil. • MAE em anos cai de forma suave.	• Lote anômalo catapulta os pesos. • Perda dispara ou vira NaN.
Solução: ReLU + Kaiming Init + BatchNorm2d	**Ação: Manter AdamW + Scheduler e logar no TensorBoard**	**Solução: clip_grad_norm_(1.0) + Reduzir Learning Rate**

Inspecionar $||\nabla L||_2$ antes de `optimizer.step()` detecta instabilidades antes que o modelo quebre.

Simulador Interativo: Fluxo de Gradientes por Camada

Inspecione como inicialização e normalização preservam o sinal da saída até a Conv1

BACKPROPAGATION CAMADA A CAMADA

Fluxo da Magnitude do Gradiente da Saída até a Entrada

Kaiming (ReLU) | Xavier (Tanh) | Vanishing (< 1) | Exploding (> 1) | Pesos Zerados

Camada 1 (Entrada RGB) ← Direção do Backpropagation (Regra da Cadeia) Camada 8 (Cabeça Linear)



Kaiming Normal (He) + BatchNorm2d: mantém o sinal equilibrado da cabeça de regressão até a Conv1.

$$\mathrm{Var}(W) = \frac{2}{n_{\text{in}}}$$

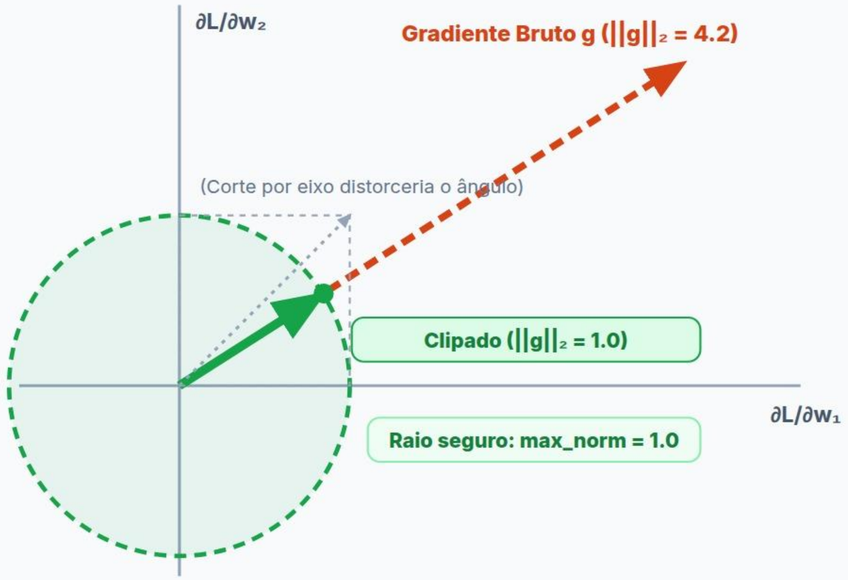
Gradient Clipping: Contendo Explosões sem Mudar a Direção

Reescalamento radial do vetor gradiente com torch.nn.utils.clip_grad_norm_

PROTEÇÃO NUMÉRICA

Geometria do Gradient Clipping por Norma Global

PRESERVA 100% DA DIREÇÃO!



REGRA DE REESCALA PROPORCIONAL

$$g \leftarrow g \cdot \frac{\text{max_norm}}{\max(\|g\|_2,\ \text{max_norm})}$$

Se $\|g\|_2 \leq 1.0$, nada muda! Se $\|g\|_2 > 1.0$, todo o vetor encolhe mantendo o mesmo ângulo.

ORDEM OBRIGATÓRIA NO TRAINING LOOP:

```
1. optimizer.zero_grad()
2. loss.backward() # Calcula gradientes
3. clip_grad_norm_(model.parameters(), 1.0)
4. optimizer.step() # Atualiza pesos seguros
```

No Experimento 2 do notebook, o clipping com `max_norm=1.0` contém picos de gradiente em LR alta sem desviar o rumo da otimização.

Painel Interativo de Diagnóstico de Treinamento

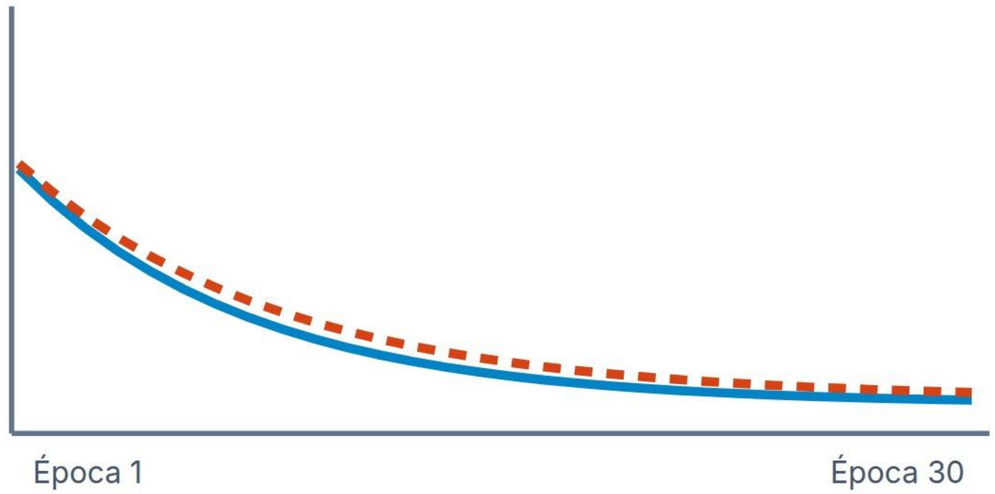
Experimente o impacto conjunto de BatchNorm2d, Dropout e Gradient Clipping

Painel Simulado de Diagnóstico (Curvas de Loss & Norma L2)

Sem Norm | **BatchNorm2d** | LayerNorm | ☑ Dropout (0.25) | ☑ Clip Norm (1.0)

Perda MSE (Treino vs. Validação)

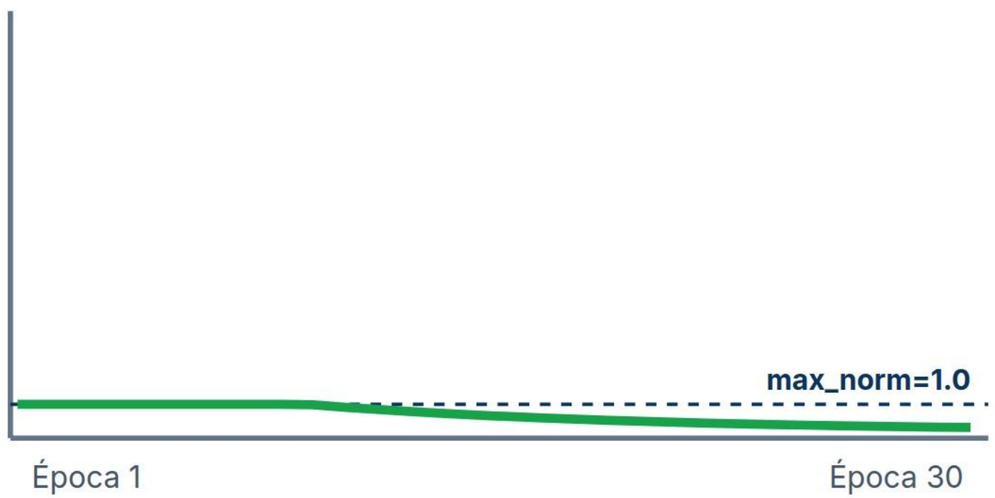
Treino: 2.3 | Val: 2.8



— MSE Treino -- MSE Validação

Norma dos Gradientes $||\nabla L||_2$

Pico: 1.00



Estado: **Gradientes Contidos e Suaves**

Gap Treino/Validação: 0.53 — Treinamento estável, regularizado e com ótima generalização.

TensorBoard na Prática: Anatomia do SummaryWriter

Como organizar subpastas em runs/ e explorar as 5 abas principais de auditoria

PROTAGONISTA DA AULA 08

Anatomia do TensorBoard (`SummaryWriter`): 5 Abas para Auditar Redes

1 SUBPASTA = 1 EXPERIMENTO COMPARÁVEL

runs/aula08_utkface/

Cada treino grava sua subpasta:

- **`00_inspecao_modelo`** — Grade de rostos + Grafo
- **`exp1_sgd / momentum / adamw`** — Duelo de otimizadores
- **`exp2_lr_alta_sem / com_clip`** — Efeito do Clipping
- **`exp3_age_cnn_principal`** — Treino completo + HParams
- **`exp4_optuna_melhor`** — Parâmetros vencedores

Os 5 Painéis do SummaryWriter na Aula 08

1. IMAGES — Auditoria Visual

```
writer.add_image("UTKFace/Lote", grid)
writer.add_figure("Real_vs_Pred", fig)
```

Confirma dados desnormalizados e dispersão.

2. GRAPHS — Topologia da Rede

```
modelo.eval()
writer.add_graph(modelo, xb)
```

Inspeciona blocos Conv2d, Pool e Linear.

3. SCALARS — Curvas em Paralelo

```
add_scalar("Metricas/Val_MAE_Anos", v, ep)
add_scalar("Diagnostico/GradNorm", g, ep)
```

Compara SGD, Momentum e AdamW no mesmo plot!

4. HISTOGRAMS — Pesos & Grads

```
add_histogram("Pesos/conv1", w, ep)
add_histogram("Gradientes/conv1", g, ep)
```

Detecta pesos mortos ou gradientes explosivos.

5. HPARAMS — Tabela de Hiperparâmetros × Métricas Finais

```
writer.add_hparams({"lr": 1.5e-3, "dropout": 0.25}, {"hparam/mae": mae}, run_name=".")
```

Cruza LR, Weight Decay, Dropout e Canais com o menor MAE obtido (inclusive do Optuna).

Com `%tensorboard --logdir runs/aula08_utkface`, todos os experimentos aparecem organizados para comparação imediata.

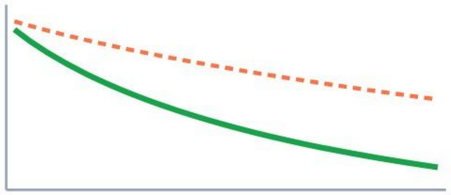
OBSERVABILIDADE & PERSISTÊNCIA

Comparação de Runs no TensorBoard + Checkpoint via state_dict

SUMMARYWRITER + WEIGHTS_ONLY=TRUE

TensorBoard — runs/aula08_utkface/exp3_age_cnn_principal

SCALARS: Val_MAE_Anos (Runs)



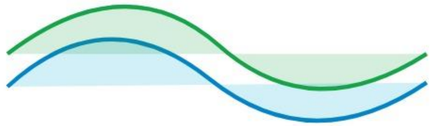
add_scalar("Metricas/Val_MAE_Anos", mae, ep)

DIAGNOSTICO: GradNorm & LR



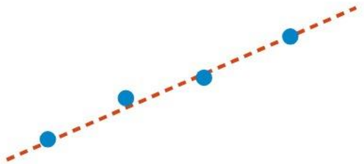
add_scalar("Diagnostico/LearningRate", lr, ep)

HISTOGRAMS: Pesos & Gradientes



writer.add_histogram("Pesos/conv1", w, ep)

IMAGES: Predicoes/Real_vs_Predito



writer.add_figure("Predicoes/Real_vs_Predito", fig)

Checkpoint Inteligente (state_dict)

1. Gatilho na Validação

```
if val_loss < best_val_loss:
```

Ignora épocas tardias com overfitting

2. Salva Dicionário de Tensores

```
torch.save(ckpt, "age_cnn_best.pt")
```

Portável e seguro (nunca salve objeto inteiro!)

3. Recarga Segura no Teste / Produção

```
m = AgeCNN().to(device)
ckpt = torch.load(path, weights_only=True)
m.load_state_dict(ckpt["model_state_dict"])
```

Enquanto o TensorBoard compara múltiplos experimentos lado a lado, o checkpoint guarda somente os pesos da época de menor erro.

Busca Bayesiana de Hiperparâmetros com Optuna

Explorando espaços contínuos com TPE, podando com MedianPruner e registrando no TensorBoard

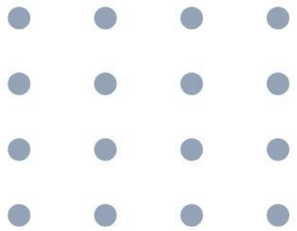
SINTONIA AUTOMATIZADA

Otimização Bayesiana com Optuna: Amostrador TPE + MedianPruner

ENCONTRA MELHORES HIPERPARÂMETROS MAIS RÁPIDO

1. Grid Search vs. Optuna (TPE Bayesiano)

Grid Search (Cego)



Gasta tempo em zonas ruins

Optuna TPE (Inteligente)



Concentra testes perto do ótimo

2. Pruning Antecipado (MedianPruner)



trial.report(val_mae, epoch) → if trial.should_prune(): raise TrialPruned()

Espaço de Busca na AgeCNN (Aula 08) → Melhor Trial logado via add_hparams():

lr	weight_decay	dropout_rate	base_channels	max_norm
[3e-4, 5e-3] (log)	[1e-5, 1e-2] (log)	[0.10, 0.40]	{8, 12, 16}	{0.5, 1.0, 2.0, 5.0}

Com TPE, poda automática e `add_hparams` no TensorBoard, o Optuna encontra ótimos hiperparâmetros em minutos.

Quiz de Fixação: Regressão Visual, Otimização & TensorBoard

Consolide os conceitos-chave antes de executar o notebook passo a passo

CONSOLIDAÇÃO DE CONHECIMENTO

Quiz Interativo: Regressão Visual, Otimização & Diagnóstico

✅ Verificar Respostas (0/4)

1. No UTKFace + TensorBoard (Aula 08), como adaptamos a CNN para regressão e comparamos experimentos?

A) Saída Linear(128, 1) sem ativação + MSELoss, gravando cada treino em subpastas de runs/aula08_utkface/.

B) Saída com Softmax de 100 classes e todos os testes sobrescrevendo o mesmo arquivo de log.

C) ReLU obrigatória na saída final para impedir qualquer erro residual durante o treino.

D) CrossEntropyLoss aplicada diretamente sobre a idade contínua extraída do arquivo.

2. Qual é a diferença central do otimizador AdamW em relação ao Adam clássico?

A) O AdamW remove completamente a estimativa de momento de 1ª ordem.

B) O AdamW desacopla o Weight Decay do passo adaptativo, penalizando os pesos de forma uniforme.

C) O AdamW só pode ser utilizado em arquiteturas MLP tabulares.

D) O AdamW substitui a função de perda MSELoss pela CrossEntropyLoss.

3. O que faz exatamente torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)?

A) Zera todas as camadas cuja norma de gradiente ultrapassar 1.0.

B) Reduz proporcionalmente todo o vetor de gradientes se $||\nabla L||_2 > 1.0$, preservando 100% da direção.

C) Congela a taxa de aprendizado em 1.0 até o final do treinamento.

D) Substitui automaticamente valores NaN pela média do batch.

4. Qual é a combinação correta para reagir a platôs e persistir o melhor modelo em produção?

A) StepLR + torch.save(model, "model.pt") no final da última época.

B) ReduceLROnPlateau monitorando val_loss + torch.save(model.state_dict(), path) quando val_loss cai.

C) Taxa fixa alta + salvar o modelo apenas quando train_loss chegar a zero.

D) Aumentar o learning rate sempre que o erro de validação parar de cair.