

Técnicas Avançadas de Redes Neurais Convolucionais

Eduardo Adame

Redes Neurais

29 de outubro de 2025





Objetivos da Aula

- **Data Augmentation:** Técnicas para expandir datasets artificialmente
- **Pipelines de Dados:** Como usar geradores de dados eficientemente
- **API Funcional do Keras:** Arquiteturas complexas e flexíveis
- **Transfer Learning:** Reutilizar conhecimento de modelos pré-treinados
- **Técnicas Avançadas:** Callbacks, regularização e otimização

Por que estas técnicas são importantes?

- **Dados limitados:** Nem sempre temos milhões de imagens rotuladas
- **Eficiência:** Treinar do zero é caro e demorado
- **Generalização:** Evitar overfitting e melhorar performance
- **Produção:** Técnicas essenciais para sistemas reais



Data Augmentation: O Problema

Desafio Principal: Escassez de Dados

- Obter dados rotulados é:
 - ▶ Caro (tempo e recursos humanos)
 - ▶ Demorado (processo manual de rotulação)
 - ▶ Difícil (alguns domínios têm poucos exemplos)
- Consequências da falta de dados:
 - ▶ Overfitting: Modelo memoriza treino, não generaliza
 - ▶ Baixa robustez: Sensível a pequenas variações
 - ▶ Viés: Não representa diversidade do mundo real

Pergunta Fundamental

Como fazer mais com menos dados?

Solução: Data Augmentation - criar variações sintéticas dos dados existentes



Data Augmentation: O Conceito

Ideia Central

- Se uma imagem de cadeira rotacionada ainda é uma cadeira...
- Se uma cadeira espelhada horizontalmente ainda é uma cadeira...
- Se uma cadeira com zoom ainda é uma cadeira...
- **Então podemos criar múltiplos exemplos de treino a partir de uma única imagem!**

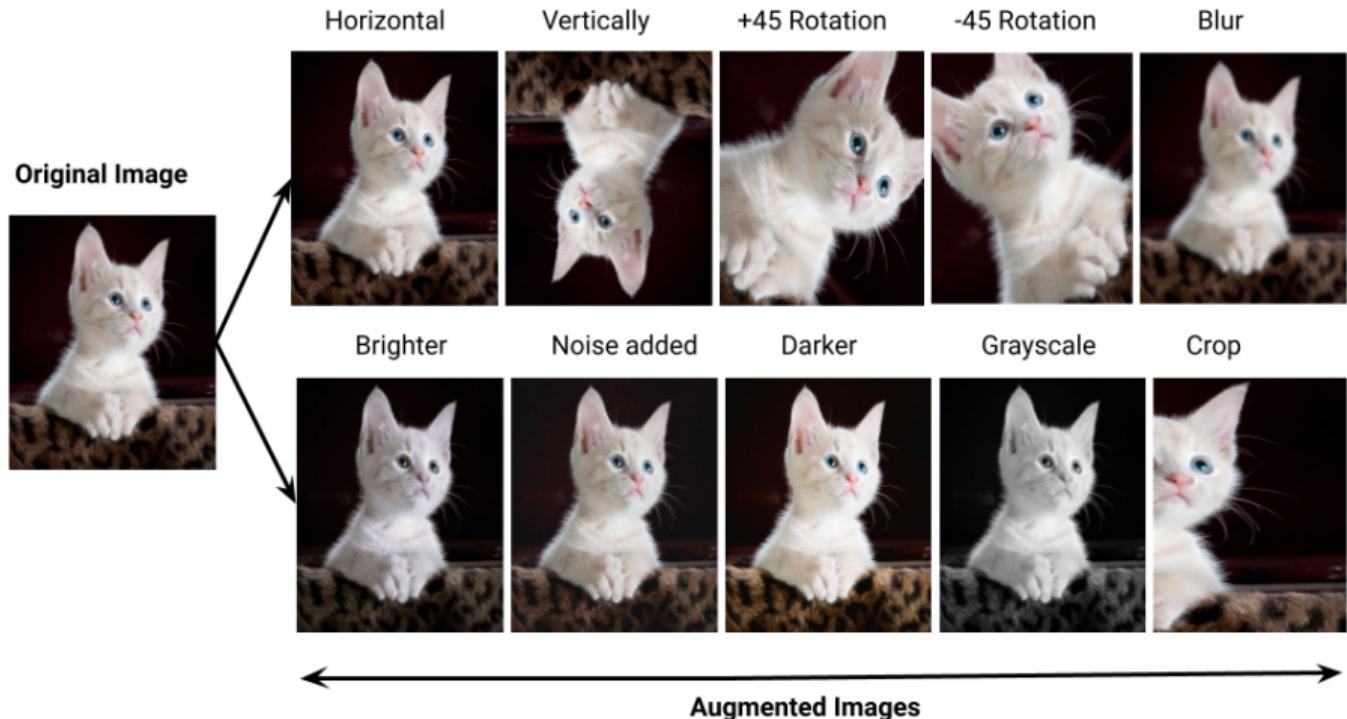
Benefícios

- **Aumenta o dataset efetivo:** De N para $N \times k$ amostras
- **Regularização implícita:** Rede aprende invariâncias úteis
- **Melhor generalização:** Exposição a variações realistas
- **Reduz overfitting:** Menos provável memorizar os dados

Cuidado!

Nem todas as transformações preservam o significado semântico!

Data Augmentation: O Conceito



Data Augmentation: Cuidados Necessários



Exemplo: Placas de Trânsito

✓ Transformações Válidas

- Pequenas rotações (± 15)
- Mudanças de brilho
- Pequenos crops
- Blur leve
- Ruído moderado

✗ Transformações Inválidas

- Flip horizontal/vertical
- Rotações grandes
- Distorções severas
- Inversão de cores
- Crops agressivos

Por que? Placa de “Proibido virar à esquerda” \neq “Proibido virar à direita”

Princípio Geral

- Aplicar apenas transformações que **preservam a classe**
- Considerar o **contexto da aplicação**
- Testar se as augmentations fazem sentido no domínio



1. Transformações Geométricas

- **Rotação:** Girar a imagem por um ângulo aleatório
- **Translação:** Deslocar horizontal/verticalmente
- **Escala/Zoom:** Ampliar ou reduzir partes da imagem
- **Flip:** Espelhamento horizontal ou vertical
- **Cisalhamento (shear):** Deformação angular

2. Transformações de Cor

- **Brilho:** Ajustar iluminação global
- **Contraste:** Aumentar/diminuir diferença entre pixels
- **Saturação:** Intensidade das cores
- **Hue:** Mudança de tonalidade
- **Normalização:** Ajuste de canais RGB



3. Transformações Baseadas em Recortes

- **Random Crop:** Recortar região aleatória da imagem
- **Center Crop:** Recortar região central
- **Cutout (2017):** Mascarar regiões aleatórias com zeros
- **Random Erasing:** Similar ao Cutout, com pixels aleatórios

4. Transformações Modernas

- **MixUp (2018):** Combinar duas imagens linearmente
 - ▶ $x' = \lambda x_1 + (1 - \lambda)x_2$
 - ▶ $y' = \lambda y_1 + (1 - \lambda)y_2$
- **CutMix (2019):** Recortar e colar patches entre imagens
- **AutoAugment (2019):** Busca automática de políticas de augmentation
- **RandAugment (2020):** Versão simplificada e mais eficiente

Data Augmentation em Keras 3 (2025)



Atualização Importante: Keras 3

- Keras 3 (lançado em 2023, estável em 2024-2025)
- API moderna e multi-backend (TensorFlow, JAX, PyTorch)
- `ImageDataGenerator` está **deprecated**
- Nova abordagem recomendada: `keras.layers.preprocessing`

Listing 1: Abordagem Moderna com Keras 3

```
1 import keras
2 from keras import layers
3 data_augmentation = keras.Sequential([
4     layers.RandomFlip("horizontal"),
5     layers.RandomRotation(0.1),  # ±10% = ±36°
6     layers.RandomZoom(0.1),
7     layers.RandomContrast(0.1),
8     layers.RandomBrightness(0.1),
9 ])
```

Pipeline Completo com tf.data



Listing 2: Pipeline Moderno de Dados

```
1 import tensorflow as tf
2 train_ds = tf.keras.utils.image_dataset_from_directory(
3     'data/train',
4     image_size=(224, 224),
5     batch_size=32,
6     label_mode='categorical'
7 )
8 AUTOTUNE = tf.data.AUTOTUNE
9 train_ds = (train_ds
10     .map(lambda x, y: (data_augmentation(x, training=True), y),
11           num_parallel_calls=AUTOTUNE)
12     .prefetch(AUTOTUNE)
13 )
14 model.fit(train_ds, epochs=50, validation_data=val_ds)
```

Vantagens: Mais eficiente, integrado ao modelo, funciona em GPU

Comparação: Abordagem Legada vs Moderna



Abordagem Antiga (`ImageDataGenerator` - deprecated)

Listing 3: Método Antigo

```
1 from keras.preprocessing.image import ImageDataGenerator
2
3 datagen = ImageDataGenerator(
4     rotation_range=20,
5     width_shift_range=0.2,
6     height_shift_range=0.2,
7     horizontal_flip=True
8 )
9
10 # Treinar com fit_generator (também deprecated)
11 model.fit_generator(
12     datagen.flow(x_train, y_train, batch_size=32),
13     epochs=50
14 )
```



Problemas:

- Executa em CPU (lento)
- API inconsistente
- Difícil integrar com tf.data

Augmentation como Camada do Modelo

Melhor Prática 2025: Augmentation dentro do Modelo



Listing 4: Integração no Modelo

```
1 # Definir augmentation
2 augmentation = keras.Sequential([
3     layers.RandomFlip("horizontal"),
4     layers.RandomRotation(0.2),
5     layers.RandomZoom(0.2),
6 ], name="data_augmentation")
7 # Criar modelo com augmentation integrado
8 inputs = keras.Input(shape=(224, 224, 3))
9 x = augmentation(inputs) # Só aplicado durante treino
10 x = layers.Rescaling(1./255)(x)
11 # Resto do modelo
12 x = layers.Conv2D(32, 3, activation='relu')(x)
13 # ... mais camadas ...
14 outputs = layers.Dense(10, activation='softmax')(x)
15 model = keras.Model(inputs, outputs)
```

Vantagens: GPU, reproduzível, exportável, treino vs inferência automático



Boas Práticas de Data Augmentation

Diretrizes Gerais

- **Comece simples:** Flip + pequenas rotações + zoom
- **Aumente gradualmente:** Adicione complexidade se necessário
- **Valide visualmente:** Inspecione exemplos augmentados
- **Considere o domínio:** Nem tudo faz sentido para todos os problemas
- **Monitore overfitting:** Se persiste, aumente augmentation

Augmentation vs Tamanho do Dataset

- **Dataset pequeno (<1000):** Augmentation agressiva é crucial
- **Dataset médio (1K-100K):** Augmentation moderada ajuda
- **Dataset grande ($>100K$):** Augmentation leve já suficiente
- **Dataset enorme ($>1M$):** Augmentation menos crítica

Cuidado: Augmentation muito agressiva pode prejudicar!



Limitações do Modelo Sequential

- Apenas topologias lineares: Uma camada após a outra
- Uma entrada, uma saída: Não suporta múltiplos inputs/outputs
- Sem compartilhamento de camadas: Cada camada usada uma vez
- Sem skip connections: Impossível fazer ResNet, U-Net, etc.

Quando Usar API Funcional?

- Múltiplas entradas: Ex: imagem + metadados
- Múltiplas saídas: Ex: classificação + regressão simultâneas
- Grafos complexos: ResNet, Inception, U-Net
- Compartilhamento de pesos: Siamese networks
- Skip connections: Qualquer arquitetura moderna

Filosofia

- Camadas são **chamáveis** (callable) em tensores
- Cada camada retorna um tensor
- Construir o grafo conectando camadas
- Especificar inputs e outputs explicitamente

Listing 5: Exemplo Simples

```
1 import keras
2 from keras import layers
3 # 1. Definir entrada
4 inputs = keras.Input(shape=(28, 28, 1))
5 # 2. Construir grafo aplicando camadas
6 x = layers.Flatten()(inputs)
7 x = layers.Dense(128, activation='relu')(x)
8 x = layers.Dropout(0.5)(x)
9 outputs = layers.Dense(10, activation='softmax')(x)
10 # 3. Criar modelo especificando inputs e outputs
11 model = keras.Model(inputs=inputs, outputs=outputs)
```

Exemplo: Múltiplas Saídas

Caso de Uso: Classificar idade e gênero simultaneamente



Listing 6: Modelo com Duas Saídas

```
1 # Entrada: imagem de rosto
2 inputs = keras.Input(shape=(224, 224, 3))
3 # Feature extraction compartilhado
4 x = layers.Conv2D(32, 3, activation='relu')(inputs)
5 x = layers.MaxPooling2D()(x)
6 x = layers.Conv2D(64, 3, activation='relu')(x)
7 x = layers.MaxPooling2D()(x)
8 x = layers.Flatten()(x)
9 x = layers.Dense(256, activation='relu')(x)
10 # Saída 1: Idade (regressão)
11 age_output = layers.Dense(1, name='age')(x)
12 # Saída 2: Gênero (classificação binária)
13 gender_output = layers.Dense(1, activation='sigmoid',
14                             name='gender')(x)
15 model = keras.Model(inputs=inputs,
16                      outputs=[age_output, gender_output])
```

Listing 7: Especificar Losses e Métricas por Saída

```
1 model.compile(  
2     optimizer='adam',  
3     loss={  
4         'age': 'mse',           # Mean Squared Error para idade  
5         'gender': 'binary_crossentropy'  # BCE para gênero  
6     },  
7     loss_weights={  
8         'age': 0.5,            # Peso relativo de cada loss  
9         'gender': 0.5  
10    },  
11    metrics={  
12        'age': ['mae'],          # Mean Absolute Error  
13        'gender': ['accuracy'] # Acurácia  
14    }  
15 )  
16  
17
```



Listing 8: Especificar Losses e Métricas por Saída

```
1 # Treinar
2 history = model.fit(
3     x_train,
4     {'age': y_age_train, 'gender': y_gender_train},
5     validation_data=(x_val, {'age': y_age_val,
6                           'gender': y_gender_val}),
7     epochs=50
8 )
```

Exemplo: Múltiplas Entradas

Caso de Uso: Classificação com imagem + metadados



Listing 9: Modelo com Duas Entradas

```
1 # Entrada 1: Imagem
2 image_input = keras.Input(shape=(224, 224, 3), name='image')
3 x1 = layers.Conv2D(32, 3, activation='relu')(image_input)
4 x1 = layers.Flatten()(x1)
5 # Entrada 2: Metadados (ex: características numéricas)
6 metadata_input = keras.Input(shape=(10,), name='metadata')
7 x2 = layers.Dense(32, activation='relu')(metadata_input)
8 # Concatenar features de ambas entradas
9 concatenated = layers.concatenate([x1, x2])
10 # Camadas de classificação final
11 x = layers.Dense(128, activation='relu')(concatenated)
12 outputs = layers.Dense(5, activation='softmax')(x)
13
14 model = keras.Model(
15     inputs=[image_input, metadata_input],
16     outputs=outputs
17 )
```

Listing 10: Bloco Residual

```
1 def residual_block(x, filters, stride=1):
2     # Salvar identidade para skip connection
3     shortcut = x
4     # Caminho principal
5     x = layers.Conv2D(filters, 3, strides=stride, padding='same')(x)
6     x = layers.BatchNormalization()(x)
7     x = layers.ReLU()(x)
8
9     x = layers.Conv2D(filters, 3, padding='same')(x)
10    x = layers.BatchNormalization()(x)
11    # Ajustar shortcut se necessário
12    if stride != 1:
13        shortcut = layers.Conv2D(filters, 1, strides=stride)(shortcut)
14        shortcut = layers.BatchNormalization()(shortcut)
15    # Adicionar skip connection
16    x = layers.Add()([x, shortcut])
17    x = layers.ReLU()(x)
18    return x
```

Listing 11: Mini-ResNet

```
1 def build_mini_resnet(input_shape, num_classes):
2     inputs = keras.Input(shape=input_shape)
3     # Stem
4     x = layers.Conv2D(64, 7, strides=2, padding='same')(inputs)
5     x = layers.BatchNormalization()(x)
6     x = layers.ReLU()(x)
7     x = layers.MaxPooling2D(3, strides=2, padding='same')(x)
8     # Blocos residuais
9     x = residual_block(x, 64)
10    x = residual_block(x, 64)
11    x = residual_block(x, 128, stride=2)
12    x = residual_block(x, 128)
13    x = residual_block(x, 256, stride=2)
14    x = residual_block(x, 256)
15    # Cabeça de classificação
16    x = layers.GlobalAveragePooling2D()(x)
17    outputs = layers.Dense(num_classes, activation='softmax')(x)
18    return keras.Model(inputs, outputs, name='mini_resnet')
```

Transfer Learning: Conceito



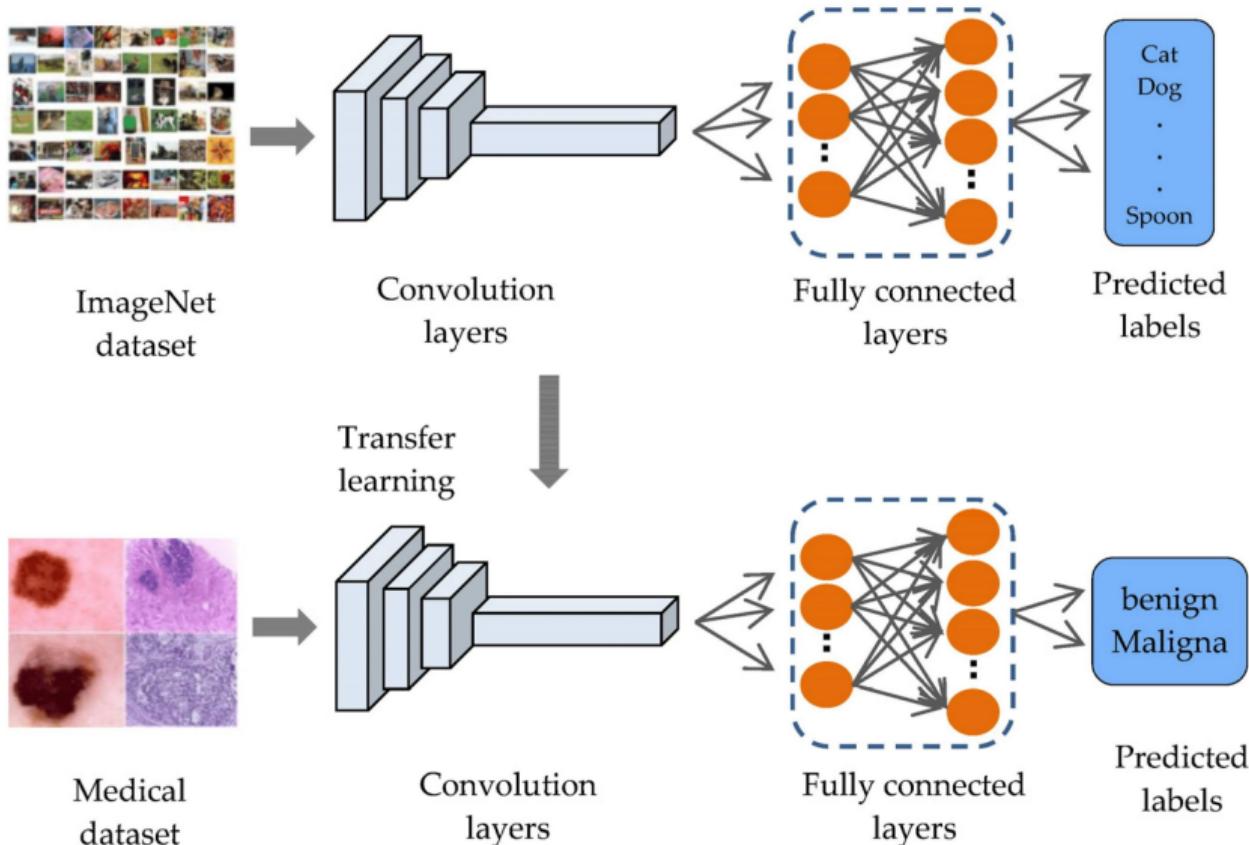
Ideia Central

- Treinar CNN do zero requer:
 - ▶ Milhões de imagens rotuladas
 - ▶ Muito poder computacional
 - ▶ Dias/semanas de treinamento
- Solução: Reutilizar conhecimento de modelos pré-treinados!

Por que funciona?

- Features iniciais são genéricas:
 - ▶ Bordas, texturas, cores
 - ▶ Padrões básicos presentes em todas as imagens
- Features finais são específicas:
 - ▶ Objetos complexos da tarefa original
 - ▶ Podem ser adaptadas para nova tarefa

Transfer Learning: O Conceito





Transfer Learning: Estratégias

1. Feature Extractor (Congelar Base)

- **Quando usar:** Dataset pequeno, domínio similar
- **Método:** Congelar camadas convolucionais, treinar apenas topo
- **Vantagem:** Rápido, menos risco de overfitting

2. Fine-tuning (Ajustar Parcialmente)

- **Quando usar:** Dataset médio/grande
- **Método:** Congelar camadas iniciais, treinar camadas finais
- **Vantagem:** Adapta features para tarefa específica

3. Treino Completo com Pesos Iniciais

- **Quando usar:** Dataset grande, domínio diferente
- **Método:** Usar pesos pré-treinados como inicialização
- **Vantagem:** Convergência mais rápida

Listing 12: Feature Extraction

```
1 # Carregar modelo pré-treinado
2 base_model = keras.applications.ResNet50(
3     weights='imagenet',           # Pesos do ImageNet
4     include_top=False,          # Remover classificador
5     input_shape=(224, 224, 3)
6 )
7 # Congelar base
8 base_model.trainable = False
9 # Adicionar novo classificador
10 inputs = keras.Input(shape=(224, 224, 3))
11 x = base_model(inputs, training=False)
12 x = layers.GlobalAveragePooling2D()(x)
13 x = layers.Dense(256, activation='relu')(x)
14 x = layers.Dropout(0.5)(x)
15 outputs = layers.Dense(10, activation='softmax')(x)
16 model = keras.Model(inputs, outputs)
```

Transfer Learning: Fine-tuning



Listing 13: Fine-tuning em Duas Etapas

```
1 # Etapa 1: Treinar apenas o topo
2 model.compile(optimizer='adam', loss='categorical_crossentropy',
3                 metrics=['accuracy'])
4 model.fit(train_ds, epochs=10, validation_data=val_ds)
5 # Etapa 2: Descongelar algumas camadas finais
6 base_model.trainable = True
7 # Congelar todas exceto últimas 20 camadas
8 for layer in base_model.layers[:-20]:
9     layer.trainable = False
10 # Recompilar com learning rate menor
11 model.compile(
12     optimizer=keras.optimizers.Adam(1e-5), # LR baixo!
13     loss='categorical_crossentropy',
14     metrics=['accuracy']
15 )
16 # Continuar treinamento
17 history = model.fit(train_ds, epochs=20, validation_data=val_ds)
```

Modelos Pré-treinados Disponíveis (2025)

Opções em keras.applications

| Modelo | Parâmetros | Top-1 Acc | Melhor Para |
|------------------|------------|-----------|----------------------|
| MobileNetV3 | 5M | 75.6% | Mobile, edge devices |
| EfficientNetB0 | 5M | 77.1% | Balance geral |
| ResNet50 | 26M | 80.4% | Baseline confiável |
| EfficientNetV2-M | 54M | 85.1% | Alta performance |
| ConvNeXt-Large | 198M | 86.6% | Estado da arte |

Considerações para Escolha

- Recursos limitados: MobileNet, EfficientNetB0
- Produção padrão: ResNet50, EfficientNetB3
- Máxima acurácia: ConvNeXt, EfficientNetV2
- Domínio específico: Buscar modelos especializados



O que são Callbacks?

- Funções chamadas em pontos específicos do treinamento
- Permitem monitorar e controlar o processo
- Essenciais para treino em produção

Listing 14: Callbacks Essenciais

```
1 from keras.callbacks import (
2     ModelCheckpoint, EarlyStopping, ReduceLROnPlateau,
3     TensorBoard, CSVLogger
4 )
5 callbacks = [
6     # Salvar melhor modelo
7     ModelCheckpoint('best_model.keras',
8                     monitor='val_accuracy',
9                     save_best_only=True),
10    # Parar se não melhorar
11    EarlyStopping(monitor='val_loss', patience=10,
12                  restore_best_weights=True),
13    # Reduzir LR se estagnar
14    ReduceLROnPlateau(monitor='val_loss', factor=0.5,
15                      patience=5, min_lr=1e-7)
16 ]
```

Callbacks: TensorBoard

Visualização Interativa do Treinamento



Listing 15: Configurar TensorBoard

```
1 import datetime
2 # Criar diretório com timestamp
3 log_dir = f"logs/fit/{datetime.datetime.now():%Y%m%d-%H%M%S}"
4 tensorboard_callback = TensorBoard(
5     log_dir=log_dir,
6     histogram_freq=1,          # Histogramas de pesos
7     write_graph=True,          # Visualizar grafo
8     write_images=True,         # Visualizar augmentations
9     update_freq='epoch',       # Frequência de atualização
10    profile_batch='10,20'      # Profiling de performance
11 )
12 # Treinar
13 model.fit(train_ds, epochs=50,
14            validation_data=val_ds,
15            callbacks=[tensorboard_callback])
16 # Visualizar: tensorboard --logdir=logs/fit
```

Callbacks Personalizados



Listing 16: Criar Callback Personalizado

```
1 class CustomCallback(keras.callbacks.Callback):
2     def on_epoch_end(self, epoch, logs=None):
3         # Executado ao fim de cada época
4         if logs['val_accuracy'] > 0.95:
5             print(f"\nAcurácia alvo atingida em {epoch}!")
6             self.model.stop_training = True
7
8     def on_train_begin(self, logs=None):
9         print("Iniciando treinamento...")
10
11    def on_train_end(self, logs=None):
12        print("Treinamento finalizado!")
13
14 # Uso
15 custom_cb = CustomCallback()
16 model.fit(train_ds, epochs=100,
17            callbacks=[custom_cb])
```

Casos de Uso: Logging personalizado, salvamento periódico, visualizações



Dropout

- **Conceito:** Desativar neurônios aleatoriamente durante treino
- **Taxa típica:** 0.2 a 0.5
- **Onde usar:** Antes de camadas Dense, após GlobalPooling
- **Efeito:** Força rede a aprender features redundantes

Batch Normalization

- **Conceito:** Normalizar ativações entre mini-batches
- **Onde usar:** Após Conv2D, antes ou depois da ativação
- **Benefícios:**
 - ▶ Acelera convergência
 - ▶ Permite learning rates maiores
 - ▶ Regularização implícita
 - ▶ Reduz sensibilidade à inicialização

Listing 17: CNN Moderna com Regularização

```
1 def build_regularized_model():
2     inputs = keras.Input(shape=(224, 224, 3))
3
4     # Stem com BatchNorm
5     x = layers.Conv2D(64, 7, strides=2, padding='same')(inputs)
6     x = layers.BatchNormalization()(x)
7     x = layers.ReLU()(x)
8
9     # Blocos com BN e Dropout
10    for filters in [64, 128, 256]:
11        x = layers.Conv2D(filters, 3, padding='same')(x)
12        x = layers.BatchNormalization()(x)
13        x = layers.ReLU()(x)
14        x = layers.MaxPooling2D()(x)
15        x = layers.Dropout(0.25)(x)  # Spatial dropout
```



Listing 18: CNN Moderna com Regularização

```
1 # Classificador com Dropout
2 x = layers.GlobalAveragePooling2D()(x)
3 x = layers.Dense(512)(x)
4 x = layers.BatchNormalization()(x)
5 x = layers.ReLU()(x)
6 x = layers.Dropout(0.5)(x)
7 outputs = layers.Dense(10, activation='softmax')(x)
8
9 return keras.Model(inputs, outputs)
```



Evolução dos Otimizadores

- **SGD**: Baseline, requer tuning cuidadoso
- **Momentum**: SGD + memória de gradientes
- **Adam (2014)**: Adaptive learning rate, popular
- **AdamW (2017)**: Adam com weight decay correto
- **Lion (2023)**: Eficiente, menos memória

Recomendações 2025

- **Primeiro experimento**: Adam com $LR=1e-3$
- **Transfer learning**: Adam com $LR=1e-5$ (fine-tuning)
- **Treino longo**: AdamW com cosine decay
- **Recursos limitados**: Lion (menor memória)
- **Máxima performance**: SGD com momentum + scheduler

Learning Rate Scheduling



Listing 19: Schedulers de Learning Rate

```
1 from keras.optimizers.schedules import (
2     ExponentialDecay, CosineDecay, PolynomialDecay
3 )
4 # 1. Decaimento Exponencial
5 lr_schedule = ExponentialDecay(
6     initial_learning_rate=1e-3,
7     decay_steps=1000,
8     decay_rate=0.96
9 )
10 # 2. Cosine Decay (popular em 2025)
11 lr_schedule = CosineDecay(
12     initial_learning_rate=1e-3,
13     decay_steps=10000,
14     alpha=0.01  # LR minimo
15 )
```



Listing 20: Schedulers de Learning Rate

```
1 # 3. Warm-up + Decay
2 def create_warmup_schedule(warmup_steps, total_steps):
3     def schedule(step):
4         if step < warmup_steps:
5             return step / warmup_steps
6         else:
7             return (total_steps - step) / (total_steps - warmup_steps)
8     return schedule
9
10 optimizer = keras.optimizers.AdamW(learning_rate=lr_schedule)
```



O que é?

- Usar float16 para aceleração, float32 para estabilidade
- **Benefícios:** 2-3x mais rápido, usa menos memória
- **Requisito:** GPU com Tensor Cores (Volta+)

Mixed Precision Training (2025)



Listing 21: Habilitar Mixed Precision

```
1 # Configurar política de mixed precision
2 keras.mixed_precision.set_global_policy('mixed_float16')
3 # Modelo normal - Keras cuida do resto automaticamente
4 model = build_model()
5 # Última camada deve ser float32 para estabilidade
6 outputs = layers.Dense(10, activation='softmax',
7                         dtype='float32', name='predictions')(x)
8 # Compilar normalmente
9 model.compile(
10     optimizer='adam',
11     loss='categorical_crossentropy',
12     metrics=['accuracy']
13 )
14 # Treinar - 2-3x mais rápido!
15 model.fit(train_ds, epochs=50)
```



1. Gradient Accumulation

- Simular batches maiores em hardware limitado
- Acumular gradientes por N steps antes de atualizar
- Útil para modelos muito grandes

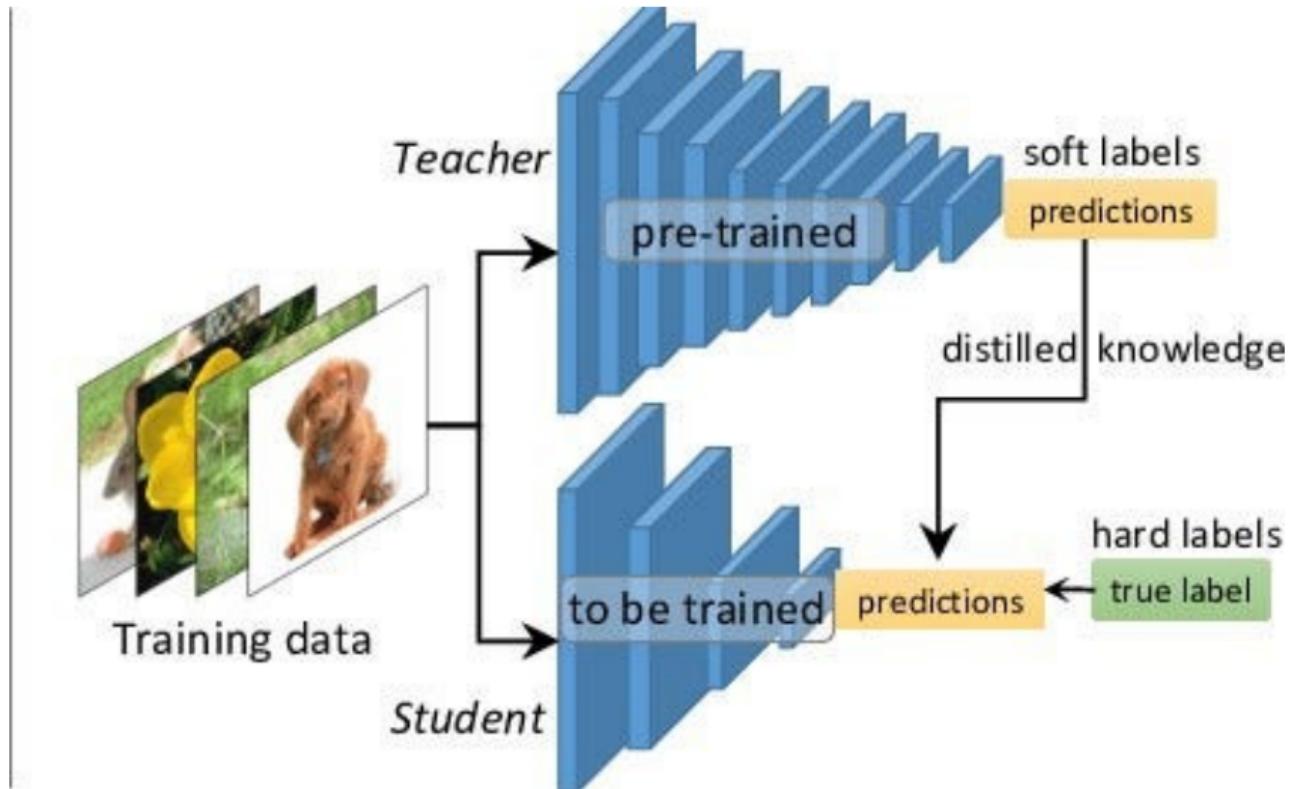
2. Stochastic Weight Averaging (SWA)

- Média de checkpoints durante final do treino
- Melhor generalização com custo mínimo
- Implementado como callback em Keras 3

3. Knowledge Distillation

- Treinar modelo pequeno (student) com modelo grande (teacher)
- Transferir conhecimento via soft targets
- Conseguir 90%+ da performance com modelo 10x menor

Knowledge Distillation





Listing 22: Debugging e Monitoramento

```
1 # 1. Verificar dataset
2 for images, labels in train_ds.take(1):
3     print(f"Batch shape: {images.shape}")
4     print(f"Label shape: {labels.shape}")
5     print(f"Min/Max: {images.numpy().min():.2f}, {images.numpy().max():.2f}")
6 # 2. Visualizar modelo
7 model.summary()
8 keras.utils.plot_model(model, show_shapes=True, to_file='model.png')
9 # 3. Gradient checking
10 with tf.GradientTape() as tape:
11     predictions = model(images, training=True)
12     loss = loss_fn(labels, predictions)
13 gradients = tape.gradient(loss, model.trainable_weights)
14 print(f"Número de gradientes: {len(gradients)}")
15 print(f"Gradiente médio: {tf.reduce_mean([tf.reduce_mean(tf.abs(g))
16             for g in gradients]):.6f}")
```

Checklist: Treinamento Efetivo



Antes de Treinar

- Dataset balanceado e representativo
- Normalização/padronização aplicada
- Data augmentation adequada ao domínio
- Train/val/test splits bem definidos
- Baseline estabelecida

Durante Treinamento

- Monitorar overfitting (train vs val loss)
- Verificar convergência do learning rate
- Usar callbacks (early stopping, checkpoints)
- Visualizar métricas no TensorBoard
- Salvar checkpoints regularmente

Checklist: Treinamento Efetivo (cont.)



Debugging

- Verificar se modelo consegue overfit em batch pequeno
- Inspecionar previsões em exemplos individuais
- Visualizar ativações e feature maps
- Checar distribuição de gradientes
- Validar pipeline de dados

Otimização

- Tunar hyperparâmetros sistematicamente
- Experimentar diferentes arquiteturas
- Testar diferentes augmentations
- Considerar ensemble de modelos
- Avaliar no conjunto de teste apenas no final



Exemplo Completo: Pipeline de Produção

Componentes de um Pipeline Robusto

1. Preparação de Dados

- ▶ tf.data.Dataset com prefetching e caching
- ▶ Augmentation integrada ao modelo
- ▶ Validação cruzada estratificada

2. Modelo

- ▶ Transfer learning com fine-tuning
- ▶ Regularização (Dropout, BatchNorm, L2)
- ▶ Mixed precision para performance

3. Treinamento

- ▶ Callbacks completos (checkpoint, early stopping, LR reduction)
- ▶ TensorBoard para monitoramento
- ▶ Salvamento de métricas e logs

4. Validação e Deploy

- ▶ Avaliação no test set
- ▶ Análise de erros
- ▶ Otimização para inferência (TFLite, ONNX)



Listing 23: Pipeline de Produção

```
1 import keras
2 from keras import layers
3 import tensorflow as tf
4 # 1. Preparar dados
5 train_ds = tf.keras.utils.image_dataset_from_directory('data/train',
6   ↪  image_size=(224,224), batch_size=32).cache().prefetch(tf.data.AUTOTUNE)
7 # 2. Augmentation
8 augmentation = keras.Sequential([
9     layers.RandomFlip(), layers.RandomRotation(0.2),
10    layers.RandomZoom(0.2), layers.RandomContrast(0.2)
11])
```

Template: Pipeline Completo (cont.)



Listing 24: Pipeline de Produção - Continuação

```
1 # 3. Transfer learning
2 base = keras.applications.EfficientNetB3(
3     include_top=False, weights='imagenet'
4 )
5 base.trainable = False
6
7 inputs = keras.Input(shape=(224, 224, 3))
8 x = augmentation(inputs)
9 x = layers.Rescaling(1./255)(x)
10 x = base(x, training=False)
11 x = layers.GlobalAveragePooling2D()(x)
12 x = layers.Dense(256, activation='relu')(x)
13 x = layers.BatchNormalization()(x)
14 x = layers.Dropout(0.5)(x)
15 outputs = layers.Dense(10, activation='softmax',
16                         dtype='float32')(x)
17 model = keras.Model(inputs, outputs)
```

Template: Pipeline Completo (cont.)



Listing 25: Pipeline de Produção - Continuação

```
1 # 4. Compilar
2 model.compile(
3     optimizer=keras.optimizers.Adam(1e-3),
4     loss='sparse_categorical_crossentropy',
5     metrics=['accuracy'])
6 )
7 # 5. Callbacks
8 callbacks = [
9     keras.callbacks.ModelCheckpoint('best.keras',
10                                     save_best_only=True),
11     keras.callbacks.EarlyStopping(patience=10,
12                                     restore_best_weights=True),
13     keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=5),
14     keras.callbacks.TensorBoard(log_dir='logs')
15 ]
16 # 6. Treinar
17 model.fit(train_ds, epochs=100, validation_data=val_ds,
18            callbacks=callbacks)
```



Recursos e Próximos Passos

Documentação Oficial

- Keras 3: keras.io - API reference completa
- TensorFlow: tensorflow.org/tutorials
- Papers: arxiv.org/list/cs.CV/recent

Práticas Recomendadas

- Começar com modelos pré-treinados
- Experimentar sistematicamente (controle de versão, MLflow)
- Investir tempo em preparação de dados
- Monitorar tudo com TensorBoard
- Validar no mundo real, não apenas métricas

Próxima Aula

- Tópico: Texto + Word Vectors

Resumo da Aula

Principais Conceitos



- Data Augmentation
 - ▶ Expandir datasets artificialmente
 - ▶ Usar camadas de preprocessing em Keras 3
 - ▶ Escolher augmentations apropriadas ao domínio

- API Funcional
 - ▶ Flexibilidade para grafos complexos
 - ▶ Múltiplas entradas/saídas
 - ▶ Skip connections e arquiteturas modernas

- Transfer Learning
 - ▶ Reutilizar conhecimento pré-treinado
 - ▶ Feature extraction vs fine-tuning
 - ▶ Escolher modelo base apropriado

- Boas Práticas
 - ▶ Callbacks para controle de treino
 - ▶ Regularização (Dropout, BatchNorm)
 - ▶ Mixed precision para performance



Q: Ainda posso usar `ImageDataGenerator`?

A: Funciona, mas está deprecated. Use `keras.layers` para augmentation.

Q: Quando usar Sequential vs Functional API?

A: Sequential para modelos simples lineares. Functional para tudo mais.

Q: Quanto data augmentation é demais?

A: Se val loss > train loss e não melhora, reduza augmentation.

Q: Sempre usar transfer learning?

A: Quase sempre! Treinar do zero só se: dataset imenso OU domínio muito diferente.

Q: Mixed precision quebra meu modelo?

A: Raro. Se acontecer, verifique overflow em loss (use loss scaling automático).

Obrigado!

Dúvidas?