Comparativo Completo de Ajustes em CNNs com Código – Explicação + Quando Usar

Dropout (Evitar overfitting)

📘 Explicação: Desativa aleatoriamente neurônios durante o treino para forçar generalização.

🕒 Quando usar: Quando a acurácia de validação para de melhorar e a de treino continua subindo.

🔹 Código original:

```
model.add(Dense(64, activation='relu'))
```

🔸 Código ajustado:

```
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
```

Data Augmentation

📘 Explicação: Cria variações artificiais das imagens de treino.

🕒 Quando usar: Quando há overfitting ou poucos dados.

🔹 Código original:

```
model.fit(x_train, y_train, epochs=10)
```

🔸 Código ajustado:

```
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
datagen.fit(x_train)
model.fit(datagen.flow(x_train, y_train), epochs=10)
```

Aumento de Épocas

📘 Explicação: Permite mais tempo de aprendizado.

🕒 Quando usar: Quando a acurácia ainda está subindo no fim do treino.

🔹 Código original:

```
model.fit(..., epochs=10)
```

🔸 Código ajustado:

```
model.fit(..., epochs=30)
```

Reduzir batch_size

📘 Explicação: Atualiza pesos com mais frequência e usa menos RAM.

🕒 Quando usar: Quando há pouco recurso de memória ou modelo instável.

🔹 Código original:

```
model.fit(..., batch_size=64)
```

🔸 Código ajustado:

```
model.fit(..., batch_size=32)
```

Mais filtros Conv2D

📘 Explicação: Extrai mais padrões por imagem.

🕒 Quando usar: Quando há underfitting (baixa acurácia e perda alta).

🔹 Código original:

```
model.add(Conv2D(32, (3, 3), activation='relu'))
```

🔸 Código ajustado:

```
model.add(Conv2D(64, (3, 3), activation='relu'))
```

Adicionar blocos Conv + Pooling

📘 Explicação: Aumenta a profundidade da rede para maior capacidade.

🕒 Quando usar: Quando a rede é rasa e não aprende bem.

🔹 Código original:

```
model.add(Conv2D(32, (3, 3), activation='relu'))
```

🔸 Código ajustado:

```
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
```

Trocar otimizador

📘 Explicação: Usa outro método de ajuste de pesos.

🕒 Quando usar: Quando o modelo não converge bem com Adam.

🔹 Código original:

```
model.compile(optimizer='adam', ...)
```

🔸 Código ajustado:

```
model.compile(optimizer='SGD', ...)
```

Mudar tamanho do kernel

📘 Explicação: Muda o campo de visão dos filtros.

🕒 Quando usar: Quando padrões precisam ser capturados em maior escala.

🔹 Código original:

```
Conv2D(64, (3, 3), activation='relu')
```

🔸 Código ajustado:

```
Conv2D(64, (5, 5), activation='relu')
```

Alterar função de ativação

📘 Explicação: Evita saturação ou inatividade dos neurônios.

🕒 Quando usar: Quando ReLU não melhora perda/resultado.

🔹 Código original:

```
Dense(64, activation='relu')
```

🔸 Código ajustado:

```
Dense(64, activation='tanh')
```

Definir learning rate manualmente

📘 Explicação: Ajusta o tamanho do passo de aprendizado.

🕒 Quando usar: Quando o modelo aprende muito devagar ou oscila.

🔹 Código original:

```
model.compile(optimizer='adam')
```

🔸 Código ajustado:

```
from tensorflow.keras.optimizers import Adam
model.compile(optimizer=Adam(learning_rate=0.0001))
```

Mais neurônios na Dense

📘 Explicação: Aumenta a capacidade da rede de classificar corretamente.

🕒 Quando usar: Quando a saída está confusa ou subótima.

🔹 Código original:

```
model.add(Dense(64, activation='relu'))
```

🔸 Código ajustado:

```
model.add(Dense(256, activation='relu'))
```

validation_split

📘 Explicação: Cria divisão automática para validação durante treino.

🕒 Quando usar: Quando não há conjunto de validação separado.

🔹 Código original:

```
model.fit(x_train, y_train, ...)
```

🔸 Código ajustado:

```
model.fit(x_train, y_train, validation_split=0.1, ...)
```

EarlyStopping

📘 Explicação: Evita overfitting interrompendo treino automaticamente.

🕒 Quando usar: Quando quer economizar tempo e evitar treino excessivo.

🔹 Código original:

```
model.fit(...)
```

🔸 Código ajustado:

```
from keras.callbacks import EarlyStopping
model.fit(..., callbacks=[EarlyStopping(patience=3)])
```

📌 Mais Ajustes Possíveis

Além dos ajustes detalhados anteriormente, aqui estão outras opções avançadas ou complementares que você pode experimentar em CNNs com Keras:

- ✔ **Batch Normalization**: Normaliza a saída de cada camada, acelerando o treinamento e aumentando a estabilidade.
  - Exemplo: `model.add(BatchNormalization())`

- ✔ **GlobalAveragePooling2D**: Substitui a etapa de Flatten + Dense, reduzindo o número de parâmetros.
  - Exemplo: `model.add(GlobalAveragePooling2D())`

- ✔ **Regularização L2/L1**: Penaliza pesos excessivos para evitar overfitting.
  - Exemplo: `Dense(64, kernel_regularizer=l2(0.001))`

- ✔ **Learning Rate Scheduler**: Altera a taxa de aprendizado durante o treinamento.
  - Exemplo: `LearningRateScheduler(lambda epoch: 1e-3 * 0.9 ** epoch)`

- ✔ **Data Augmentation avançado (Albumentations)**: Biblioteca com transformações mais poderosas que o Keras nativo.

- ✔ **Redes pré-treinadas (Transfer Learning)**: Usar modelos como VGG, ResNet ou EfficientNet com pesos do ImageNet.

- ✔ **Ajuste fino (Fine-Tuning)**: Descongelar parte das camadas de uma rede pré-treinada e continuar o treinamento.

- ✔ **Redução de dimensionalidade com PCA ou Autoencoders**: para pré-processamento antes da CNN.

- ✔ **Ajuste de arquitetura com Keras Tuner**: ferramenta para experimentar combinações automáticas de hiperparâmetros.