# Engenharia de Redes Neurais Artificiais

**Aluno: Matheus Paul Lopuch**

**Exercícios das Aulas**

**Aula 03_CNN**

**Arquiteturas Especializadas**

**em Visão Computacional**

**Exercícios:**

1. **Verifique o exemplo da aula 3 para visão computacional.**
2. **Porque a EfficientNet-Lite0 foi menos eficiente que a LeNet5?**
3. **Como podemos melhorar a resultado da EfficientNet-Lite0? Implemente esta solução.**

**Respostas:**

1. **O exemplo da aula 3 trata de arquiteturas de redes neurais convolucionais (CNNs) aplicadas à visão computacional.**

2. **A menor eficiência da EfficientNet-Lite0, nesse contexto, pode se dar por alguns motivos:**
   **- Overfitting ou Underfitting: Uma arquitetura muito grande pode não generalizar bem em conjuntos de dados pequenos ou específicos.**
   **- Hiperparâmetros/desempenho: A EfficientNet-Lite0 depende de uma boa configuração de hiperparâmetros para ser eficiente, enquanto a LeNet-5 já é otimizada para MNIST.**
   **- Capacidade computacional: Modelos maiores exigem mais recurso de hardware e acabam sendo penalizados na eficiência quando comparados a modelos pequenos e eficazes para tarefas simples.**

3. **Ajuste dos hiperparâmetros: Aprimorar taxa de aprendizado, tamanho do batch e número de epochs.**
   **Data augmentation: Aplicar técnicas para aumentar o número e a variação dos exemplos de treino.**
   **Regularização: Uso de Dropout e Batch Normalization.**
   **Early stopping: Evita overfitting interrompendo o treinamento quando a validação não melhora mais.**

Segue abaixo o código completo para melhorar os resultados da EfficientNet-Lite0 utilizando data augmentation, regularização (Dropout), ajuste de hiperparâmetros e callbacks (EarlyStopping) em TensorFlow/Keras. (Substituir NUM_CLASSES pela quantidade de classes do seu problema e x_train, y_train, x_val, y_val estejam no formato e dimensão adequados).

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.layers import Dropout, Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Carregar dados
# x_train, y_train, x_val, y_val devem estar preparados (imagens já padronizadas para o EfficientNet)
# Por exemplo, usando:
# x_train = tf.image.resize(x_train, (224, 224))

# Modelo EfficientNet-Lite0 (substitua por EfficientNetB0 se não houver Lite)
base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(224,224,3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)                    # Regularização com Dropout
predictions = Dense(NUM_CLASSES, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
```

```
# Compilar
model.compile(optimizer=tf.keras.optimizers.Adam(learning_
rate=1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Callback para EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', patience=7,
restore_best_weights=True)

# Treinamento
model.fit(
    datagen.flow(x_train, y_train, batch_size=32),
    epochs=50,
    validation_data=(x_val, y_val),
    callbacks=[early_stop]
)

# Avaliação
loss, acc = model.evaluate(x_val, y_val)
print("Val Loss:", loss, "| Val Accuracy:", acc)
```