Disciplina:

Análise de Imagem e Visão Computacional

Professor: Octavio Santana

PUC Minas

Aula 4

Modelos Pré-Treinados e Fine-Tuning

Professor: Octavio Santana

PUC Minas

Objetivos da Aula

Ao final desta aula, os alunos serão capazes de:

1. Utilização de redes pré treinadas como VGG, MobileNet, ResNet
2. Estratégias de fine-tuning e transfer learning

PUC Minas

Introdução a modelos pré-treinados

O que são modelos pré-treinados

Redes neurais pré-treinadas são modelos de Deep Learning que foram treinados em grandes conjuntos de dados (como ImageNet, COCO, ou OpenImages) e estão disponíveis publicamente para reutilização. Esses modelos já aprenderam features hierárquicas (bordas, texturas, formas, objetos) que são frequentemente transferíveis para outras tarefas de visão computacional.

Vantagens:

- Eficiência computacional
  - Evita treinar uma rede do zero, economizando tempo e recursos.
- Bom desempenho com pouco dado
  - Útil quando o dataset disponível é pequeno.
- Generalização aprimorada
  - Modelos treinados em ImageNet já capturam padrões visuais genéricos.

Arquiteturas principais de redes Pré-Treinadas

VGG (VGG16 e VGG19):

- Desenvolvida pela Oxford's Visual Geometry Group (VGG) em 2014.
- Características:
  - Estrutura simples: Sequência de blocos de convolução (3x3) + max-pooling.
  - 16 ou 19 camadas (VGG16 e VGG19).
  - Pesos grandes (~500MB) devido a camadas totalmente conectadas (FC).
- Indicação: Boa para transfer learning em tarefas onde simplicidade é prioridade.

ResNet (ResNet50, ResNet101, ResNet152):

- Introduziu Residual Connections (Skip Connections) para resolver o problema de vanishing gradient em redes profundas.
- Características:
  - Blocos residuais permitem treinar redes com 50, 101 ou até 152 camadas.
  - Mais eficiente que VGG (menos parâmetros, melhor generalização).
  - Versátil: Usado em classificação, detecção de objetos e segmentação.
- Indicação: Uma das melhores escolhas para fine-tuning em problemas gerais.

Arquiteturas principais de redes Pré-Treinadas

EfficientNet (B0 a B7):

- Proposta pelo Google em 2019, otimiza escala composta (depth, width, resolution).
- Características:
  - Mais leve e eficiente que ResNet e VGG.
  - AutoML-based: Arquitetura otimizada automaticamente.
  - B0 (menor) a B7 (maior) – Escalonável conforme necessidade computacional.
- Indicação: Ideal para aplicações com restrições de hardware (mobile, edge devices).

MobileNet (V1, V2, V3):

- Projetada para dispositivos móveis e embarcados.
- Características:
  - Convoluções separáveis (Depthwise Separable Convolutions) reduzem drasticamente os parâmetros.
  - MobileNetV3 inclui otimizações com Neural Architecture Search (NAS).
- Indicação: Aplicações em tempo real (IoT, smartphones).

Arquiteturas principais de redes Pré-Treinadas

Tabela comparativa

Modelo	Parâmetros (milhões)	Tamanho (MB)	Melhor uso
VGG16	138	~528	Baseline, estudos iniciais
ResNet50	25,5	~98	Fine-tuning geral
EfficientNetB0	5,3	~20	Aplicações eficientes
MobileNetV3	4,2	~16	Dispositivos móveis

Transfer Learning

Transfer Learning

O que é Transfer Learning?

- Técnica que aproveita o conhecimento de um modelo pré-treinado em uma nova tarefa.

Quando usar?

- Quando o dataset novo é pequeno (evitar overfitting).
- Quando a nova tarefa é similar à original (ex: classificação de animais → classificação de raças de cachorro).

Transfer Learning

Abordagens comuns:

- Extração de Features (Feature Extraction)
  - Congelar todas as camadas convolucionais.
  - Adicionar novas camadas densas no final para a nova tarefa.
  - Treinar apenas as novas camadas.
- Fine-tuning (Ajuste Fino)
  - Descongelar algumas camadas convolucionais finais.
  - Re-treinar parcialmente o modelo com uma taxa de aprendizado baixa.

Extração de Features (Feature Extraction)

Extração de Features (Feature Extraction)

Técnica que utiliza as camadas convolucionais de um modelo pré-treinado como extrator de características, congelando todos os pesos e treinando apenas um novo classificador no topo da rede.

Passos para Extração de Features:

Carregar Modelo Pré-Treinado

```
from tensorflow.keras.applications import ResNet50

base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,224,3))
```

Congelar todas as camadas do modelo

```
base_model.trainable = False  # Todas as camadas ficam não-treináveis
```

Extração de Features (Feature Extraction)

Adicionar novas camadas para classificação

```
from tensorflow.keras import layers, Model

inputs = tf.keras.Input(shape=(224,224,3))
x = base_model(inputs, training=False)  # Modo inferência
x = layers.GlobalAveragePooling2D()(x)
outputs = layers.Dense(num_classes, activation='softmax')(x)
model = Model(inputs, outputs)
```

Compilar e treinar apenas as novas camadas

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_generator, epochs=20, validation_data=val_generator)
```

Extração de Features (Feature Extraction)

Vantagens da Extração de Features

- Computacionalmente eficiente (apenas treina poucas camadas)
- Evita overfitting em datasets pequenos
- Preserva features genéricas aprendidas no dataset original

```
base_model = ResNet50(weights="imagenet")

feature_extractor = Model(
    inputs=base_model.input,
    outputs=base_model.get_layer('avg_pool').output
)
```

```
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (224, 224))

x = np.expand_dims(img, axis=0)

pred = feature_extractor.predict(x, verbose=False)
print(pred.shape) # (1, 2048)
```

Aplicações Práticas

- Sistemas de recomendação visual:
  - Extrair features para similaridade de imagens
- Classificação com poucos dados:
  - Usar features como input para SVM/Random Forest
- Embeddings para busca:
  - Armazenar vetores de features em bancos vetoriais

PUC Minas

Fine-Tuning

Fine-Tuning

Técnica que ajusta um modelo pré-treinado para uma nova tarefa específica, re-treinando parcialmente suas camadas. Combina:

- Conhecimento geral (features aprendidas no dataset original)
- Especialização (adaptação ao novo problema)

Por que Fine-tuning Funciona?

- Transferência Hierárquica de Features:
- Camadas iniciais detectam bordas/texturas (genéricas)
- Camadas médias capturam padrões complexos (formas)
- Camadas finais identificam conceitos semânticos (específicos ao dataset original)

Dicas importantes:

- Usar augmentation de dados para evitar overfitting.
- Learning rate pequeno (ex: 1e-4 a 1e-5) para ajustes sutis.
- Monitorar validação para evitar degradação do modelo.

Passos para o Fine-Tuning

Carrega o modelo pré-treinado

```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50

# Carrega o modelo pré-treinado sem as camadas finais (include_top=False)
base_model = ResNet50(weights='imagenet',
                      include_top=False,
                      input_shape=(224, 224, 3))
```

Congela o modelo base

```
# Congela todas as camadas para o treino inicial
base_model.trainable = False
```

Passo para o Fine-Tuning

Adicionar novas camadas

```
from tensorflow.keras import layers, models

# Cria um novo modelo sobre a base congelada
inputs = tf.keras.Input(shape=(224, 224, 3))
x = base_model(inputs)
x = layers.GlobalAveragePooling2D()(x)  # Reduz para 1D
outputs = layers.Dense(10, activation='softmax')(x)  # 10 classes

model = models.Model(inputs, outputs)
```

Treinar apenas as novas camadas

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Treina apenas o novo head
model.fit(train_images, train_labels,
          epochs=5,
          validation_data=(val_images, val_labels))
```

PUC Minas

Passos para o Fine-Tuning

Descongelar e Ajustar (Fine-Tuning)

```
# Descongela as últimas 10 camadas
for layer in base_model.layers[-10:]:
    layer.trainable = True

# Recompila com learning rate menor
model.compile(optimizer=tf.keras.optimizers.Adam(1e-5),  # LR pequeno
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Treina novamente
model.fit(train_images, train_labels,
          epochs=10,
          validation_data=(val_images, val_labels))
```

Conclusões

- Nesta aula, os alunos aprenderam a utilizar redes neurais convolucionais pré-treinadas, como VGG, MobileNet e ResNet.
- Entenderam a importância dessas arquiteturas para acelerar o desenvolvimento de soluções em visão computacional.
- Foram apresentadas as estratégias de transfer learning e fine-tuning, que permitem adaptar redes já treinadas a novos contextos e conjuntos de dados.