Assignment II

AdvNNs October 31, 2025

Abstract

The goal is to review the ResNet model and its implementation in Pytorch. Please upload your solutions in one compressed file to Classroom before november 11th.

Problema 1

Considere la arquitectura AlexNet que usó con el conjunto de datos de [1]. Haga un script en PyTorch en el que aumente el número de capas ocultas y resuelva nuevamente el problema. Haga un gráfico de cómo cambian las precisiones (train y test) a medida que aumenta el número de capas ocultas. Intente encontrar un punto donde la precisión se vea comprometida si se agregan más capas ocultas. Escriba sus hallazgos.

Source: https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-small

In [1]:
```
import os
import kagglehub
import glob

path = kagglehub.dataset_download("paramaggarwal/fashion-product-images-small")

imagenes_dir = os.path.join(path, "images")

# Buscar todas las imágenes jpg en el directorio
imagenes = glob.glob(os.path.join(imagenes_dir, "*.jpg"))

# Mostrar cuántas imágenes se encontraron y ejemplo de rutas
print(f"Total de imágenes encontradas: {len(imagenes)}")
```

```
Total de imágenes encontradas: 44441
```

In [2]:
```
import torch
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from collections import OrderedDict
```

In [ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

class AlexNetVariable(nn.Module):
    """
    AlexNet con número variable de capas fully connected.
    Args:
        num_classes: número de clases de salida
        num_hidden_layers: número de capas ocultas en el clasificador (1-10)
        dropout_p: probabilidad de dropout
    """
    def __init__(self, num_classes=1000, num_hidden_layers=2, dropout_p=0.5):
        super(AlexNetVariable, self).__init__()

        self.num_hidden_layers = num_hidden_layers

        # Feature extractor (igual que AlexNet original)
        self.features = nn.Sequential(
            OrderedDict([
                ('conv1', nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=0)),
                ('relu1', nn.ReLU(inplace=True)),
                ('maxpool1', nn.MaxPool2d(kernel_size=3, stride=2)),
                ('conv2', nn.Conv2d(96, 256, kernel_size=5, stride=1, padding=2)),
                ('relu2', nn.ReLU(inplace=True)),
                ('maxpool2', nn.MaxPool2d(kernel_size=3, stride=2)),
                ('conv3', nn.Conv2d(256, 384, kernel_size=3, stride=1, padding=1)),
                ('relu3', nn.ReLU(inplace=True)),
                ('conv4', nn.Conv2d(384, 384, kernel_size=3, stride=1, padding=1)),
                ('relu4', nn.ReLU(inplace=True)),
                ('conv5', nn.Conv2d(384, 256, kernel_size=3, stride=1, padding=1)),
                ('relu5', nn.ReLU(inplace=True)),
                ('maxpool5', nn.MaxPool2d(kernel_size=3, stride=2)),
            ])
        )

        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))

        # Construir clasificador dinámicamente según num_hidden_layers
        classifier_layers = []

        # Primera capa: de features a primera hidden
        in_features = 256 * 6 * 6  # 9216
        hidden_size = 4096

        for i in range(num_hidden_layers):
            classifier_layers.append((f'dropout{i+1}', nn.Dropout(p=dropout_p)))
            classifier_layers.append((f'fc{i+1}', nn.Linear(in_features, hidden_siz
            classifier_layers.append((f'relu{i+1}', nn.ReLU(inplace=True)))
            in_features = hidden_size  # La siguiente capa recibe hidden_size

        # Última capa: de última hidden a num_classes
        classifier_layers.append((f'dropout_final', nn.Dropout(p=dropout_p)))
        classifier_layers.append(('fc_out', nn.Linear(hidden_size, num_classes)))

        self.classifier = nn.Sequential(OrderedDict(classifier_layers))

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

```
# Prueba rápida
test_model = AlexNetVariable(num_classes=10, num_hidden_layers=2).to(device)
print(f"Modelo con 2 capas ocultas:")
print(f"Total de parámetros: {sum(p.numel() for p in test_model.parameters()):,}")
print(f"\nArquitectura del clasificador:")
print(test_model.classifier)
```

```
Modelo con 2 capas ocultas:
Total de parámetros: 58,322,314

Arquitectura del clasificador:
Sequential(
  (dropout1): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=9216, out_features=4096, bias=True)
  (relu1): ReLU(inplace=True)
  (dropout2): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=4096, out_features=4096, bias=True)
  (relu2): ReLU(inplace=True)
  (dropout_final): Dropout(p=0.5, inplace=False)
  (fc_out): Linear(in_features=4096, out_features=10, bias=True)
)
```

In [4]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

Out[4]:
```
device(type='cuda')
```

In [ ]:
```
import pandas as pd
import os
import torch
from torch.utils.data import Dataset, DataLoader, random_split
from torchvision import transforms, models
from PIL import Image
import torch.nn as nn
import torch.optim as optim

# 1. Cargar las etiquetas
csv_path = os.path.join(path, "styles.csv")
# Usamos on_bad_lines='skip' porque este CSV en particular tiene algunos errores
df = pd.read_csv(csv_path, on_bad_lines='skip')

# Nos interesan el 'id' de la imagen y su 'articleType' (la clase)
df = df[['id', 'articleType']].copy()
df['id'] = df['id'].astype(str) + ".jpg" # Convertir ID a nombre de archivo (ej. 15
df = df.dropna() # Eliminar filas sin datos

# 2. Procesar etiquetas: Convertir texto a números

df['label_idx'], unique_classes = pd.factorize(df['articleType'])
num_classes = len(unique_classes)

print(f"Total de clases encontradas: {num_classes}")
print(f"Ejemplo de mapeo: '{unique_classes[0]}' -> 0")

# Crear un diccionario (mapa) de id_imagen -> label_idx para acceso rápido
```

```
label_map = dict(zip(df['id'], df['label_idx']))


image_files_with_labels = []
image_labels = []

for img_path in imagenes:
    img_name = os.path.basename(img_path)
    if img_name in label_map:
        image_files_with_labels.append(img_path)
        image_labels.append(label_map[img_name])

print(f"Imágenes encontradas en disco: {len(imagenes)}")
print(f"Imágenes con etiqueta válida: {len(image_files_with_labels)}")
```

```
Total de clases encontradas: 143
Ejemplo de mapeo: 'Shirts' -> 0
Imágenes encontradas en disco: 44441
Imágenes con etiqueta válida: 44419
```

In [6]:
```
import numpy as np

labels_np = np.array(image_labels)
print("Valores únicos de etiquetas:", np.unique(labels_np))
print("Mínimo:", labels_np.min(), "Máximo:", labels_np.max())
print("num_classes:", num_classes)

# Corregir num_classes si es necesario
num_classes = int(labels_np.max()) + 1
print("num_classes corregido:", num_classes)

# Chequeo de rango
if labels_np.min() < 0 or labels_np.max() >= num_classes:
    print("¡Advertencia! Hay etiquetas fuera del rango válido para CrossEntropyLoss
```

```
Valores únicos de etiquetas: [  0   1   2   3   4   5   6   7   8   9  10  11  12  1
3  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 142]
Mínimo: 0 Máximo: 142
num_classes: 143
num_classes corregido: 143
```

In [7]:
```
from tqdm import tqdm
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, random_split
from PIL import Image
import torchvision.transforms as T
import os
import matplotlib.pyplot as plt
```

```python
import numpy as np

# Configuración
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
path = "./"
num_classes = int(labels_np.max()) + 1 # Ajustar según tus etiquetas

# Dataset
class ListDataset(Dataset):
    def __init__(self, file_paths, labels, transform=None):
        self.file_paths = list(file_paths)
        self.labels = list(labels)
        self.transform = transform
    def __len__(self):
        return len(self.file_paths)
    def __getitem__(self, idx):
        img_path = self.file_paths[idx]
        label = int(self.labels[idx])
        img = Image.open(img_path).convert('RGB')
        if self.transform:
            img = self.transform(img)
        return img, label

# Transformaciones
transform = T.Compose([
    T.Resize((224, 224)),
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Dataset completo
dataset_full = ListDataset(image_files_with_labels, image_labels, transform=transfo
print(f"Tamaño del dataset válido: {len(dataset_full)}")

# Split train/val
n_train = int(0.8 * len(dataset_full))
n_val = len(dataset_full) - n_train
train_ds, val_ds = random_split(dataset_full, [n_train, n_val])

train_loader = DataLoader(train_ds, batch_size=32, shuffle=True, num_workers=0)
val_loader = DataLoader(val_ds, batch_size=64, shuffle=False, num_workers=0)
```

```
Tamaño del dataset válido: 44419
```

In [ ]:

```python
import os
import pickle
os.environ['CUDA_LAUNCH_BLOCKING'] = '1'

# Configuración del experimento
num_classes = int(labels_np.max()) + 1
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Rango de capas ocultas a probar
hidden_layers_range = [1, 2, 3, 4, 5, 6, 8, 10]

# Early Stopping Config (igual para todos)
```

```
max_epochs = 100  # Máximo permitido
patience = 5     # Épocas sin mejora antes de parar
min_delta = 0.001 # Mejora mínima considerada significativa

# Almacenar resultados
results = {
    'num_layers': [],
    'train_acc': [],
    'val_acc': [],
    'train_loss': [],
    'val_loss': [],
    'num_params': [],
    'epochs_trained': []
}

print(f"{'='*70}")
print(f"EXPERIMENTO: Efecto del número de capas ocultas en AlexNet")
print(f"{'='*70}")
print(f"Dataset: {len(dataset_full)} imágenes, {num_classes} clases")
print(f"Early Stopping: Máx {max_epochs} épocas, paciencia={patience}, min_delta={m
print(f"Learning Rate: 1e-4 (fijo, sin scheduler)")
print(f"Capas ocultas a probar: {hidden_layers_range}")
print(f"{'='*70}\n")

# Iterar sobre diferentes números de capas ocultas
for num_hidden in hidden_layers_range:
    print(f"\n{'='*60}")
    print(f"EXPERIMENTO {len(results['num_layers']) + 1}: {num_hidden} capas oculta
    print(f"{'='*60}")

    # Crear modelo
    model = AlexNetVariable(
        num_classes=num_classes,
        num_hidden_layers=num_hidden,
        dropout_p=0.5
    ).to(device)

    num_params = sum(p.numel() for p in model.parameters())
    print(f"Parámetros del modelo: {num_params:,}")

    # Optimizador y criterio
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-4)  # lr fijo para comparación

    # Listas para tracking
    train_losses_epoch = []
    val_losses_epoch = []
    train_accs_epoch = []
    val_accs_epoch = []

    # Early Stopping variables
    best_val_acc = 0.0
    best_epoch = 0
    epochs_no_improve = 0
    early_stopped = False
```

```
# Entrenamiento con Early Stopping
for epoch in range(max_epochs):
    # TRAIN
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    loop = tqdm(train_loader, desc=f"Epoch {epoch+1}/{max_epochs} [Train]", lea
    for images, labels in loop:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)
        _, preds = outputs.max(1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)

        loop.set_postfix(loss=running_loss/total, acc=correct/total)

    epoch_train_loss = running_loss / total
    epoch_train_acc = correct / total
    train_losses_epoch.append(epoch_train_loss)
    train_accs_epoch.append(epoch_train_acc)

    # VALIDATION
    model.eval()
    val_running_loss = 0.0
    val_correct = 0
    val_total = 0

    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)

            val_running_loss += loss.item() * images.size(0)
            _, preds = outputs.max(1)
            val_correct += (preds == labels).sum().item()
            val_total += labels.size(0)

    epoch_val_loss = val_running_loss / val_total
    epoch_val_acc = val_correct / val_total
    val_losses_epoch.append(epoch_val_loss)
    val_accs_epoch.append(epoch_val_acc)

    # Early Stopping Logic
    if epoch_val_acc > best_val_acc + min_delta:
        best_val_acc = epoch_val_acc
        best_epoch = epoch + 1
        epochs_no_improve = 0
```

```
            # Guardar mejor modelo
            best_model_state = model.state_dict().copy()
        else:
            epochs_no_improve += 1

        if (epoch + 1) % 5 == 0 or epoch == 0:
            print(f"  Epoch {epoch+1}: train_loss={epoch_train_loss:.4f}, train_acc
                  f"val_loss={epoch_val_loss:.4f}, val_acc={epoch_val_acc:.4f} "
                  f"[Best: {best_val_acc:.4f} @ Epoch {best_epoch}]")

        # Detener si no hay mejora
        if epochs_no_improve >= patience:
            print(f"\n⚠️  Early Stopping: No mejora en {patience} épocas. Mejor épo
            early_stopped = True
            break

    # Restaurar mejor modelo
    if early_stopped:
        model.load_state_dict(best_model_state)
        print(f"✓ Restaurado modelo de época {best_epoch} (Val Acc: {best_val_acc:.

    # Guardar modelo
    model_path = f'alexnet_{num_hidden}layers.pth'
    torch.save(model.state_dict(), model_path)
    print(f"\n✓ Modelo guardado: {model_path}")

    # Guardar resultados FINALES (del mejor modelo)
    results['num_layers'].append(num_hidden)
    results['train_acc'].append(train_accs_epoch[best_epoch-1])
    results['val_acc'].append(best_val_acc)
    results['train_loss'].append(train_losses_epoch[best_epoch-1])
    results['val_loss'].append(val_losses_epoch[best_epoch-1])
    results['num_params'].append(num_params)
    results['epochs_trained'].append(best_epoch)

    print(f"✓ Épocas entrenadas: {best_epoch}/{max_epochs}")
    print(f"✓ Best Train Accuracy: {train_accs_epoch[best_epoch-1]:.4f}")
    print(f"✓ Best Val Accuracy: {best_val_acc:.4f}")

# Guardar resultados completos
with open('experiment_results.pkl', 'wb') as f:
    pickle.dump(results, f)

print(f"\n{'='*60}")
print("EXPERIMENTO COMPLETADO")
print(f"{'='*60}")
print(f"Resultados guardados en: experiment_results.pkl")
print(f"Modelos guardados: {len(results['num_layers'])} archivos .pth")
```

```
======================================================================
EXPERIMENTO: Efecto del número de capas ocultas en AlexNet
======================================================================
Dataset: 44419 imágenes, 143 clases
Early Stopping: Máx 100 épocas, paciencia=5, min_delta=0.001
Learning Rate: 1e-4 (fijo, sin scheduler)
Capas ocultas a probar: [1, 2, 3, 4, 5, 6, 8, 10]
======================================================================


============================================================
EXPERIMENTO 1: 1 capas ocultas
============================================================
Parámetros del modelo: 42,085,903
Parámetros del modelo: 42,085,903

  Epoch 1: train_loss=1.7088, train_acc=0.5652, val_loss=0.9799, val_acc=0.7193 [Bes
t: 0.7193 @ Epoch 1]

  Epoch 5: train_loss=0.5001, train_acc=0.8414, val_loss=0.5426, val_acc=0.8309 [Bes
t: 0.8309 @ Epoch 5]

  Epoch 10: train_loss=0.2540, train_acc=0.9099, val_loss=0.5384, val_acc=0.8507 [Be
st: 0.8539 @ Epoch 9]

⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 9
✓ Restaurado modelo de época 9 (Val Acc: 0.8539)

✓ Modelo guardado: alexnet_1layers.pth
✓ Épocas entrenadas: 9/100
✓ Best Train Accuracy: 0.8985
✓ Best Val Accuracy: 0.8539

============================================================
EXPERIMENTO 2: 2 capas ocultas
============================================================

✓ Modelo guardado: alexnet_1layers.pth
✓ Épocas entrenadas: 9/100
✓ Best Train Accuracy: 0.8985
✓ Best Val Accuracy: 0.8539

============================================================
EXPERIMENTO 2: 2 capas ocultas
============================================================
Parámetros del modelo: 58,867,215
Parámetros del modelo: 58,867,215

  Epoch 1: train_loss=1.9279, train_acc=0.5061, val_loss=1.0617, val_acc=0.6934 [Bes
t: 0.6934 @ Epoch 1]

  Epoch 5: train_loss=0.5675, train_acc=0.8224, val_loss=0.6013, val_acc=0.8208 [Bes
t: 0.8208 @ Epoch 5]

```

```
  Epoch 10: train_loss=0.3127, train_acc=0.8938, val_loss=0.5155, val_acc=0.8476 [Be
st: 0.8505 @ Epoch 9]

⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 9
✓ Restaurado modelo de época 9 (Val Acc: 0.8505)

✓ Modelo guardado: alexnet_2layers.pth
✓ Épocas entrenadas: 9/100
✓ Best Train Accuracy: 0.8822
✓ Best Val Accuracy: 0.8505

============================================================
EXPERIMENTO 3: 3 capas ocultas
============================================================

✓ Modelo guardado: alexnet_2layers.pth
✓ Épocas entrenadas: 9/100
✓ Best Train Accuracy: 0.8822
✓ Best Val Accuracy: 0.8505

============================================================
EXPERIMENTO 3: 3 capas ocultas
============================================================
Parámetros del modelo: 75,648,527
Parámetros del modelo: 75,648,527

  Epoch 1: train_loss=2.1950, train_acc=0.4409, val_loss=1.2250, val_acc=0.6536 [Bes
t: 0.6536 @ Epoch 1]

  Epoch 5: train_loss=0.6278, train_acc=0.8068, val_loss=0.5900, val_acc=0.8198 [Bes
t: 0.8198 @ Epoch 5]

  Epoch 10: train_loss=0.3583, train_acc=0.8842, val_loss=0.5515, val_acc=0.8366 [Be
st: 0.8366 @ Epoch 10]

  Epoch 15: train_loss=0.2151, train_acc=0.9283, val_loss=0.6384, val_acc=0.8561 [Be
st: 0.8561 @ Epoch 15]
```

```
  Epoch 20: train_loss=0.1496, train_acc=0.9515, val_loss=0.6582, val_acc=0.8506 [Be
st: 0.8561 @ Epoch 15]

⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 15
✓ Restaurado modelo de época 15 (Val Acc: 0.8561)

✓ Modelo guardado: alexnet_3layers.pth
✓ Épocas entrenadas: 15/100
✓ Best Train Accuracy: 0.9283
✓ Best Val Accuracy: 0.8561

============================================================
EXPERIMENTO 4: 4 capas ocultas
============================================================

✓ Modelo guardado: alexnet_3layers.pth
✓ Épocas entrenadas: 15/100
✓ Best Train Accuracy: 0.9283
✓ Best Val Accuracy: 0.8561

============================================================
EXPERIMENTO 4: 4 capas ocultas
============================================================
Parámetros del modelo: 92,429,839
Parámetros del modelo: 92,429,839

  Epoch 1: train_loss=2.5135, train_acc=0.3562, val_loss=1.6103, val_acc=0.5583 [Bes
t: 0.5583 @ Epoch 1]

  Epoch 5: train_loss=0.6728, train_acc=0.7973, val_loss=0.6457, val_acc=0.8041 [Bes
t: 0.8041 @ Epoch 5]

  Epoch 10: train_loss=0.3998, train_acc=0.8719, val_loss=0.5844, val_acc=0.8323 [Be
st: 0.8323 @ Epoch 10]

  Epoch 15: train_loss=0.2649, train_acc=0.9131, val_loss=0.6215, val_acc=0.8368 [Be
st: 0.8416 @ Epoch 11]
```

```
⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 11
✓ Restaurado modelo de época 11 (Val Acc: 0.8416)

✓ Modelo guardado: alexnet_4layers.pth
✓ Épocas entrenadas: 11/100
✓ Best Train Accuracy: 0.8827
✓ Best Val Accuracy: 0.8416

============================================================
EXPERIMENTO 5: 5 capas ocultas
============================================================

✓ Modelo guardado: alexnet_4layers.pth
✓ Épocas entrenadas: 11/100
✓ Best Train Accuracy: 0.8827
✓ Best Val Accuracy: 0.8416

============================================================
EXPERIMENTO 5: 5 capas ocultas
============================================================
Parámetros del modelo: 109,211,151
Parámetros del modelo: 109,211,151

  Epoch 1: train_loss=2.7494, train_acc=0.2926, val_loss=1.7271, val_acc=0.5179 [Bes
t: 0.5179 @ Epoch 1]

  Epoch 5: train_loss=0.7966, train_acc=0.7632, val_loss=0.7461, val_acc=0.7747 [Bes
t: 0.7747 @ Epoch 5]

  Epoch 10: train_loss=0.4808, train_acc=0.8497, val_loss=0.6110, val_acc=0.8208 [Be
st: 0.8222 @ Epoch 9]

  Epoch 15: train_loss=0.3225, train_acc=0.8952, val_loss=0.6487, val_acc=0.8381 [Be
st: 0.8411 @ Epoch 14]

  Epoch 20: train_loss=0.2346, train_acc=0.9257, val_loss=0.7100, val_acc=0.8423 [Be
st: 0.8437 @ Epoch 19]

  Epoch 25: train_loss=0.2037, train_acc=0.9374, val_loss=0.7853, val_acc=0.8425 [Be
st: 0.8471 @ Epoch 24]
```

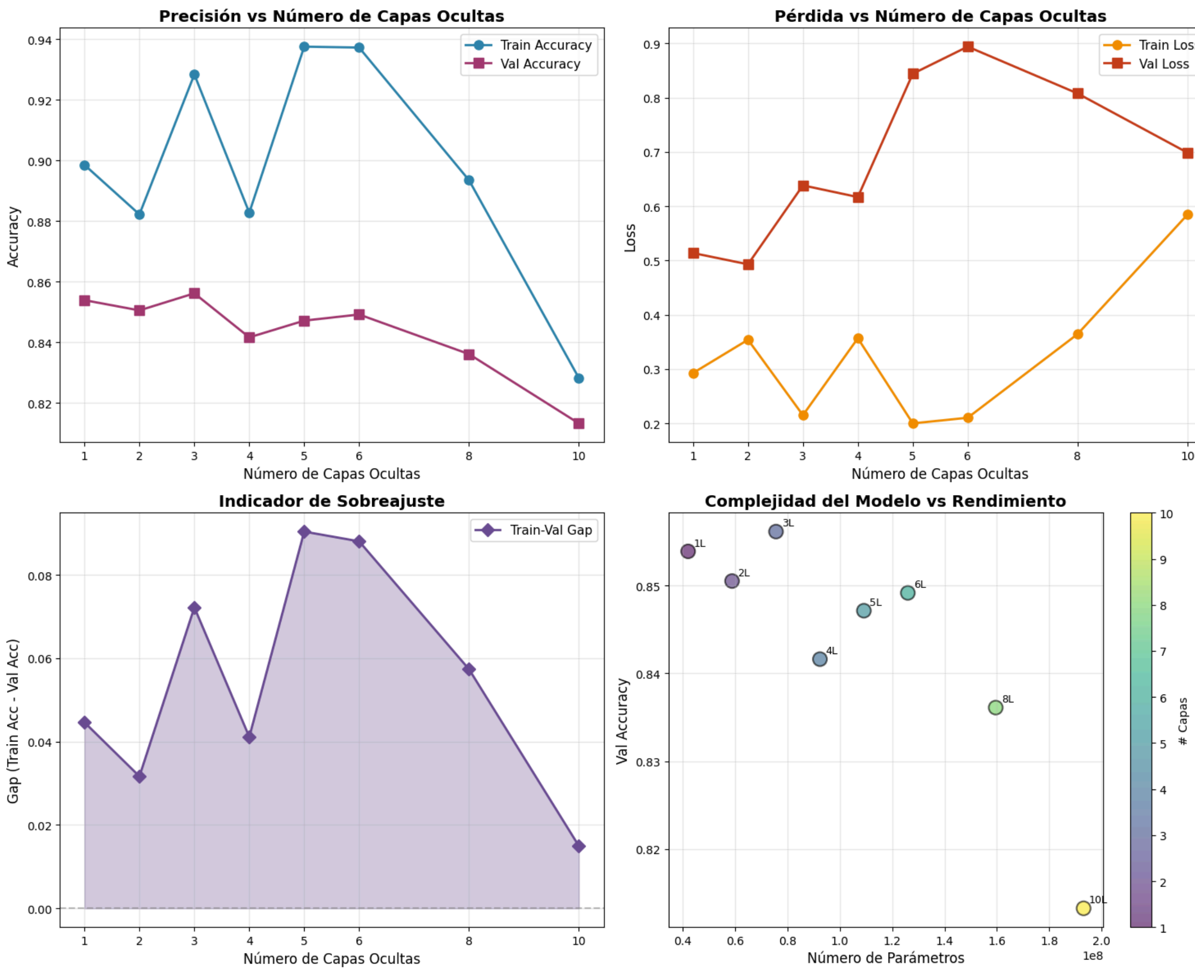
```
⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 24
✓ Restaurado modelo de época 24 (Val Acc: 0.8471)

✓ Modelo guardado: alexnet_5layers.pth
✓ Épocas entrenadas: 24/100
✓ Best Train Accuracy: 0.9375
✓ Best Val Accuracy: 0.8471

============================================================
EXPERIMENTO 6: 6 capas ocultas
============================================================

✓ Modelo guardado: alexnet_5layers.pth
✓ Épocas entrenadas: 24/100
✓ Best Train Accuracy: 0.9375
✓ Best Val Accuracy: 0.8471

============================================================
EXPERIMENTO 6: 6 capas ocultas
============================================================
Parámetros del modelo: 125,992,463
Parámetros del modelo: 125,992,463

  Epoch 1: train_loss=2.9526, train_acc=0.2424, val_loss=1.9623, val_acc=0.4678 [Bes
t: 0.4678 @ Epoch 1]

  Epoch 5: train_loss=0.8765, train_acc=0.7385, val_loss=0.7941, val_acc=0.7616 [Bes
t: 0.7616 @ Epoch 5]

  Epoch 10: train_loss=0.5492, train_acc=0.8295, val_loss=0.6499, val_acc=0.8136 [Be
st: 0.8136 @ Epoch 10]

  Epoch 15: train_loss=0.3928, train_acc=0.8779, val_loss=0.6492, val_acc=0.8362 [Be
st: 0.8362 @ Epoch 15]

  Epoch 20: train_loss=0.2891, train_acc=0.9083, val_loss=0.7007, val_acc=0.8380 [Be
st: 0.8380 @ Epoch 20]

  Epoch 25: train_loss=0.2372, train_acc=0.9278, val_loss=0.8362, val_acc=0.8456 [Be
st: 0.8456 @ Epoch 25]

  Epoch 30: train_loss=0.2178, train_acc=0.9369, val_loss=0.8644, val_acc=0.8446 [Be
st: 0.8492 @ Epoch 29]
```

```
⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 29
✓ Restaurado modelo de época 29 (Val Acc: 0.8492)

✓ Modelo guardado: alexnet_6layers.pth
✓ Épocas entrenadas: 29/100
✓ Best Train Accuracy: 0.9372
✓ Best Val Accuracy: 0.8492

============================================================
EXPERIMENTO 7: 8 capas ocultas
============================================================

✓ Modelo guardado: alexnet_6layers.pth
✓ Épocas entrenadas: 29/100
✓ Best Train Accuracy: 0.9372
✓ Best Val Accuracy: 0.8492

============================================================
EXPERIMENTO 7: 8 capas ocultas
============================================================
Parámetros del modelo: 159,555,087
Parámetros del modelo: 159,555,087

  Epoch 1: train_loss=3.0321, train_acc=0.2263, val_loss=2.1047, val_acc=0.3969 [Bes
t: 0.3969 @ Epoch 1]

  Epoch 5: train_loss=0.9940, train_acc=0.7102, val_loss=0.9397, val_acc=0.7235 [Bes
t: 0.7235 @ Epoch 5]

  Epoch 10: train_loss=0.6736, train_acc=0.7985, val_loss=0.7747, val_acc=0.7734 [Be
st: 0.7871 @ Epoch 9]

  Epoch 15: train_loss=0.5237, train_acc=0.8416, val_loss=0.6745, val_acc=0.8130 [Be
st: 0.8130 @ Epoch 15]

  Epoch 20: train_loss=0.4204, train_acc=0.8727, val_loss=0.7890, val_acc=0.8271 [Be
st: 0.8271 @ Epoch 20]

  Epoch 25: train_loss=0.3556, train_acc=0.8937, val_loss=0.8105, val_acc=0.8286 [Be
st: 0.8345 @ Epoch 23]

  Epoch 30: train_loss=0.3239, train_acc=0.9016, val_loss=0.7346, val_acc=0.8193 [Be
st: 0.8361 @ Epoch 27]
```

```
⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 27
✓ Restaurado modelo de época 27 (Val Acc: 0.8361)

✓ Modelo guardado: alexnet_8layers.pth
✓ Épocas entrenadas: 27/100
✓ Best Train Accuracy: 0.8935
✓ Best Val Accuracy: 0.8361

============================================================
EXPERIMENTO 8: 10 capas ocultas
============================================================

✓ Modelo guardado: alexnet_8layers.pth
✓ Épocas entrenadas: 27/100
✓ Best Train Accuracy: 0.8935
✓ Best Val Accuracy: 0.8361

============================================================
EXPERIMENTO 8: 10 capas ocultas
============================================================
Parámetros del modelo: 193,117,711
Parámetros del modelo: 193,117,711

  Epoch 1: train_loss=3.1453, train_acc=0.2076, val_loss=2.3540, val_acc=0.3394 [Bes
t: 0.3394 @ Epoch 1]

  Epoch 5: train_loss=1.1677, train_acc=0.6708, val_loss=1.0280, val_acc=0.7111 [Bes
t: 0.7111 @ Epoch 5]

  Epoch 10: train_loss=0.8046, train_acc=0.7630, val_loss=0.8355, val_acc=0.7539 [Be
st: 0.7539 @ Epoch 10]

  Epoch 15: train_loss=0.6741, train_acc=0.8017, val_loss=0.7086, val_acc=0.7981 [Be
st: 0.7981 @ Epoch 15]

  Epoch 20: train_loss=0.5844, train_acc=0.8282, val_loss=0.6986, val_acc=0.8133 [Be
st: 0.8133 @ Epoch 20]
```

```
  Epoch 25: train_loss=0.5563, train_acc=0.8381, val_loss=0.7669, val_acc=0.7976 [Be
st: 0.8133 @ Epoch 20]

⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 20
✓ Restaurado modelo de época 20 (Val Acc: 0.8133)

✓ Modelo guardado: alexnet_10layers.pth
✓ Épocas entrenadas: 20/100
✓ Best Train Accuracy: 0.8282
✓ Best Val Accuracy: 0.8133

============================================================
EXPERIMENTO COMPLETADO
============================================================
Resultados guardados en: experiment_results.pkl
Modelos guardados: 8 archivos .pth

✓ Modelo guardado: alexnet_10layers.pth
✓ Épocas entrenadas: 20/100
✓ Best Train Accuracy: 0.8282
✓ Best Val Accuracy: 0.8133

============================================================
EXPERIMENTO COMPLETADO
============================================================
Resultados guardados en: experiment_results.pkl
Modelos guardados: 8 archivos .pth
```

In [9]:
```python
# Cargar resultados
import pickle
with open('experiment_results.pkl', 'rb') as f:
    results = pickle.load(f)

# Crear visualizaciones
fig, axes = plt.subplots(2, 2, figsize=(15, 12))

# 1. Accuracy vs Número de capas
axes[0, 0].plot(results['num_layers'], results['train_acc'], marker='o', linewidth=
                markersize=8, label='Train Accuracy', color='#2E86AB')
axes[0, 0].plot(results['num_layers'], results['val_acc'], marker='s', linewidth=2,
                markersize=8, label='Val Accuracy', color='#A23B72')
axes[0, 0].set_xlabel('Número de Capas Ocultas', fontsize=12)
axes[0, 0].set_ylabel('Accuracy', fontsize=12)
axes[0, 0].set_title('Precisión vs Número de Capas Ocultas', fontsize=14, fontweigh
axes[0, 0].legend(fontsize=11)
axes[0, 0].grid(True, alpha=0.3)
axes[0, 0].set_xticks(results['num_layers'])

# 2. Loss vs Número de capas
axes[0, 1].plot(results['num_layers'], results['train_loss'], marker='o', linewidth
                markersize=8, label='Train Loss', color='#F18F01')
axes[0, 1].plot(results['num_layers'], results['val_loss'], marker='s', linewidth=2
                markersize=8, label='Val Loss', color='#C73E1D')
axes[0, 1].set_xlabel('Número de Capas Ocultas', fontsize=12)
axes[0, 1].set_ylabel('Loss', fontsize=12)
axes[0, 1].set_title('Pérdida vs Número de Capas Ocultas', fontsize=14, fontweight=
axes[0, 1].legend(fontsize=11)
axes[0, 1].grid(True, alpha=0.3)
axes[0, 1].set_xticks(results['num_layers'])

# 3. Gap (Overfitting indicator)
gap = np.array(results['train_acc']) - np.array(results['val_acc'])
axes[1, 0].plot(results['num_layers'], gap, marker='D', linewidth=2,
                markersize=8, color='#6A4C93', label='Train-Val Gap')
axes[1, 0].axhline(y=0, color='gray', linestyle='--', alpha=0.5)
axes[1, 0].fill_between(results['num_layers'], gap, 0, alpha=0.3, color='#6A4C93')
axes[1, 0].set_xlabel('Número de Capas Ocultas', fontsize=12)
axes[1, 0].set_ylabel('Gap (Train Acc - Val Acc)', fontsize=12)
axes[1, 0].set_title('Indicador de Sobreajuste', fontsize=14, fontweight='bold')
axes[1, 0].legend(fontsize=11)
axes[1, 0].grid(True, alpha=0.3)
axes[1, 0].set_xticks(results['num_layers'])

# 4. Número de parámetros vs Accuracy
axes[1, 1].scatter(results['num_params'], results['val_acc'], s=150, alpha=0.6,
                   c=results['num_layers'], cmap='viridis', edgecolors='black', lin
for i, num_layers in enumerate(results['num_layers']):
    axes[1, 1].annotate(f'{num_layers}L',
                        (results['num_params'][i], results['val_acc'][i]),
                        xytext=(5, 5), textcoords='offset points', fontsize=9)
axes[1, 1].set_xlabel('Número de Parámetros', fontsize=12)
axes[1, 1].set_ylabel('Val Accuracy', fontsize=12)
axes[1, 1].set_title('Complejidad del Modelo vs Rendimiento', fontsize=14, fontweig
axes[1, 1].grid(True, alpha=0.3)
cbar = plt.colorbar(axes[1, 1].collections[0], ax=axes[1, 1])
cbar.set_label('# Capas', fontsize=10)

plt.tight_layout()
plt.savefig('alexnet_depth_experiment.png', dpi=300, bbox_inches='tight')
plt.show()

print("\n" + "="*60)
print("RESUMEN DE RESULTADOS")
print("="*60)
print(f"{'Capas':<8} {'Train Acc':<12} {'Val Acc':<12} {'Gap':<10} {'Parámetros':<1
print("-"*60)
for i in range(len(results['num_layers'])):
    gap_val = results['train_acc'][i] - results['val_acc'][i]
    print(f"{results['num_layers'][i]:<8} {results['train_acc'][i]:<12.4f} "
          f"{results['val_acc'][i]:<12.4f} {gap_val:<10.4f} {results['num_params'][

# Encontrar el punto óptimo
best_val_idx = np.argmax(results['val_acc'])
print("\n" + "="*60)
print("HALLAZGOS")
print("="*60)
print(f"✓ Mejor Val Accuracy: {results['val_acc'][best_val_idx]:.4f} "
      f"con {results['num_layers'][best_val_idx]} capas ocultas")
print(f"✓ Punto de compromiso: ", end="")

# Detectar punto de compromiso (cuando val_acc empieza a bajar)
for i in range(1, len(results['val_acc'])):
    if results['val_acc'][i] < results['val_acc'][i-1]:
        print(f"A partir de {results['num_layers'][i]} capas, la precisión se degra
        break
else:
    print("No se observó degradación en el rango probado")

print(f"\n✓ Gráfico guardado: alexnet_depth_experiment.png")
```



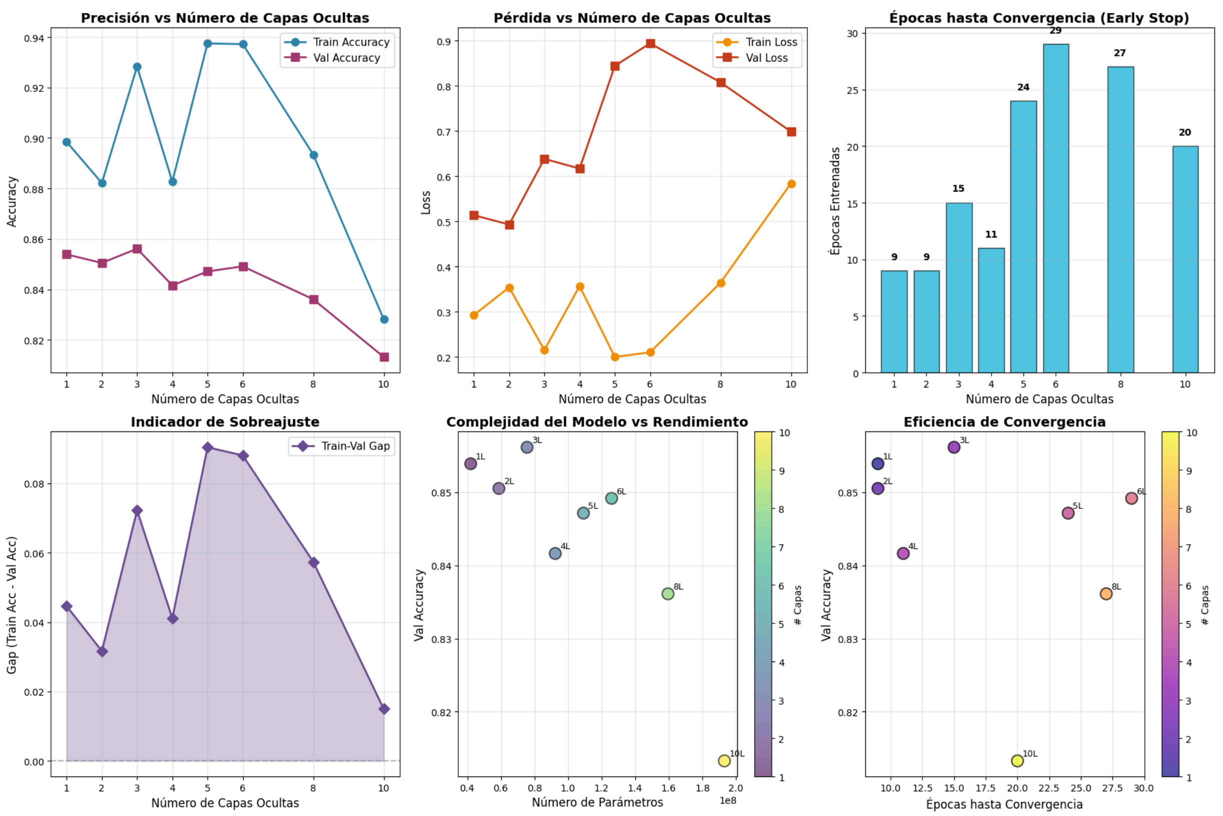
```
============================================================
RESUMEN DE RESULTADOS
============================================================
Capas    Train Acc     Val Acc      Gap         Parámetros
------------------------------------------------------------
1        0.8985        0.8539       0.0446      42,085,903
2        0.8822        0.8505       0.0317      58,867,215
3        0.9283        0.8561       0.0722      75,648,527
4        0.8827        0.8416       0.0411      92,429,839
5        0.9375        0.8471       0.0904      109,211,151
6        0.9372        0.8492       0.0880      125,992,463
8        0.8935        0.8361       0.0573      159,555,087
10       0.8282        0.8133       0.0150      193,117,711

============================================================
HALLAZGOS
============================================================
✓ Mejor Val Accuracy: 0.8561 con 3 capas ocultas
✓ Punto de compromiso: A partir de 2 capas, la precisión se degrada

✓ Gráfico guardado: alexnet_depth_experiment.png
```

In [10]:

```python
# Cargar resultados
import pickle
import matplotlib.pyplot as plt
import numpy as np

with open('experiment_results.pkl', 'rb') as f:
    results = pickle.load(f)

# Crear visualización AMPLIADA con epochs_trained
fig, axes = plt.subplots(2, 3, figsize=(18, 12))

# 1. Accuracy vs Número de capas
axes[0, 0].plot(results['num_layers'], results['train_acc'], marker='o', linewidth=
                markersize=8, label='Train Accuracy', color='#2E86AB')
axes[0, 0].plot(results['num_layers'], results['val_acc'], marker='s', linewidth=2,
                markersize=8, label='Val Accuracy', color='#A23B72')
axes[0, 0].set_xlabel('Número de Capas Ocultas', fontsize=12)
axes[0, 0].set_ylabel('Accuracy', fontsize=12)
axes[0, 0].set_title('Precisión vs Número de Capas Ocultas', fontsize=14, fontweigh
axes[0, 0].legend(fontsize=11)
axes[0, 0].grid(True, alpha=0.3)
axes[0, 0].set_xticks(results['num_layers'])

# 2. Loss vs Número de capas
axes[0, 1].plot(results['num_layers'], results['train_loss'], marker='o', linewidth
                markersize=8, label='Train Loss', color='#F18F01')
axes[0, 1].plot(results['num_layers'], results['val_loss'], marker='s', linewidth=2
                markersize=8, label='Val Loss', color='#C73E1D')
axes[0, 1].set_xlabel('Número de Capas Ocultas', fontsize=12)
axes[0, 1].set_ylabel('Loss', fontsize=12)
axes[0, 1].set_title('Pérdida vs Número de Capas Ocultas', fontsize=14, fontweight=
axes[0, 1].legend(fontsize=11)
axes[0, 1].grid(True, alpha=0.3)
axes[0, 1].set_xticks(results['num_layers'])
```

```python
# 3. NUEVO: Épocas hasta convergencia
axes[0, 2].bar(results['num_layers'], results['epochs_trained'], color='#06AED5', a
axes[0, 2].set_xlabel('Número de Capas Ocultas', fontsize=12)
axes[0, 2].set_ylabel('Épocas Entrenadas', fontsize=12)
axes[0, 2].set_title('Épocas hasta Convergencia (Early Stop)', fontsize=14, fontwei
axes[0, 2].grid(True, alpha=0.3, axis='y')
axes[0, 2].set_xticks(results['num_layers'])
for i, (x, y) in enumerate(zip(results['num_layers'], results['epochs_trained'])):
    axes[0, 2].text(x, y + 1, str(y), ha='center', fontsize=10, fontweight='bold')

# 4. Gap (Overfitting indicator)
gap = np.array(results['train_acc']) - np.array(results['val_acc'])
axes[1, 0].plot(results['num_layers'], gap, marker='D', linewidth=2,
                markersize=8, color='#6A4C93', label='Train-Val Gap')
axes[1, 0].axhline(y=0, color='gray', linestyle='--', alpha=0.5)
axes[1, 0].fill_between(results['num_layers'], gap, 0, alpha=0.3, color='#6A4C93')
axes[1, 0].set_xlabel('Número de Capas Ocultas', fontsize=12)
axes[1, 0].set_ylabel('Gap (Train Acc - Val Acc)', fontsize=12)
axes[1, 0].set_title('Indicador de Sobreajuste', fontsize=14, fontweight='bold')
axes[1, 0].legend(fontsize=11)
axes[1, 0].grid(True, alpha=0.3)
axes[1, 0].set_xticks(results['num_layers'])

# 5. Número de parámetros vs Accuracy
axes[1, 1].scatter(results['num_params'], results['val_acc'], s=150, alpha=0.6,
                   c=results['num_layers'], cmap='viridis', edgecolors='black', lin
for i, num_layers in enumerate(results['num_layers']):
    axes[1, 1].annotate(f'{num_layers}L',
                        (results['num_params'][i], results['val_acc'][i]),
                        xytext=(5, 5), textcoords='offset points', fontsize=9)
axes[1, 1].set_xlabel('Número de Parámetros', fontsize=12)
axes[1, 1].set_ylabel('Val Accuracy', fontsize=12)
axes[1, 1].set_title('Complejidad del Modelo vs Rendimiento', fontsize=14, fontweig
axes[1, 1].grid(True, alpha=0.3)
cbar = plt.colorbar(axes[1, 1].collections[0], ax=axes[1, 1])
cbar.set_label('# Capas', fontsize=10)

# 6. NUEVO: Convergencia vs Val Accuracy
axes[1, 2].scatter(results['epochs_trained'], results['val_acc'], s=150, alpha=0.7,
                   c=results['num_layers'], cmap='plasma', edgecolors='black', line
for i, num_layers in enumerate(results['num_layers']):
    axes[1, 2].annotate(f'{num_layers}L',
                        (results['epochs_trained'][i], results['val_acc'][i]),
                        xytext=(5, 5), textcoords='offset points', fontsize=9)
axes[1, 2].set_xlabel('Épocas hasta Convergencia', fontsize=12)
axes[1, 2].set_ylabel('Val Accuracy', fontsize=12)
axes[1, 2].set_title('Eficiencia de Convergencia', fontsize=14, fontweight='bold')
axes[1, 2].grid(True, alpha=0.3)
cbar2 = plt.colorbar(axes[1, 2].collections[0], ax=axes[1, 2])
cbar2.set_label('# Capas', fontsize=10)

plt.tight_layout()
plt.savefig('alexnet_depth_experiment_with_early_stopping.png', dpi=300, bbox_inche
plt.show()
```

```
print("\n" + "="*70)
print("RESUMEN DE RESULTADOS CON EARLY STOPPING")
print("="*70)
print(f"{'Capas':<8} {'Épocas':<10} {'Train Acc':<12} {'Val Acc':<12} {'Gap':<10} {
print("-"*70)
for i in range(len(results['num_layers'])):
    gap_val = results['train_acc'][i] - results['val_acc'][i]
    print(f"{results['num_layers'][i]:<8} {results['epochs_trained'][i]:<10} "
          f"{results['train_acc'][i]:<12.4f} {results['val_acc'][i]:<12.4f} "
          f"{gap_val:<10.4f} {results['num_params'][i]:<15,}")

# Análisis de convergencia
print("\n" + "="*70)
print("HALLAZGOS CON EARLY STOPPING")
print("="*70)

best_val_idx = np.argmax(results['val_acc'])
print(f"✓ Mejor Val Accuracy: {results['val_acc'][best_val_idx]:.4f} "
      f"con {results['num_layers'][best_val_idx]} capas ocultas "
      f"(convergió en {results['epochs_trained'][best_val_idx]} épocas)")

# Modelos que necesitaron más épocas
max_epochs_idx = np.argmax(results['epochs_trained'])
print(f"\n✓ Mayor tiempo de convergencia: {results['num_layers'][max_epochs_idx]} c
      f"({results['epochs_trained'][max_epochs_idx]} épocas)")
print(f"  → Modelos más profundos requieren MÁS épocas para converger")

# Modelos que convergieron rápido
min_epochs_idx = np.argmin(results['epochs_trained'])
print(f"\n✓ Convergencia más rápida: {results['num_layers'][min_epochs_idx]} capas
      f"({results['epochs_trained'][min_epochs_idx]} épocas)")

# Detectar punto de compromiso
for i in range(1, len(results['val_acc'])):
    if results['val_acc'][i] < results['val_acc'][i-1]:
        print(f"\n✓ Punto de degradación: A partir de {results['num_layers'][i]} ca
        break
else:
    print("\n✓ No se observó degradación en el rango probado")

print(f"\n✓ Gráfico guardado: alexnet_depth_experiment_with_early_stopping.png")
print("="*70)
```



```
======================================================================
RESUMEN DE RESULTADOS CON EARLY STOPPING
======================================================================
Capas    Épocas     Train Acc    Val Acc      Gap        Params
----------------------------------------------------------------------
1        9          0.8985       0.8539       0.0446     42,085,903
2        9          0.8822       0.8505       0.0317     58,867,215
3        15         0.9283       0.8561       0.0722     75,648,527
4        11         0.8827       0.8416       0.0411     92,429,839
5        24         0.9375       0.8471       0.0904     109,211,151
6        29         0.9372       0.8492       0.0880     125,992,463
8        27         0.8935       0.8361       0.0573     159,555,087
10       20         0.8282       0.8133       0.0150     193,117,711

======================================================================
HALLAZGOS CON EARLY STOPPING
======================================================================
✓ Mejor Val Accuracy: 0.8561 con 3 capas ocultas (convergió en 15 épocas)

✓ Mayor tiempo de convergencia: 6 capas (29 épocas)
  → Modelos más profundos requieren MÁS épocas para converger

✓ Convergencia más rápida: 1 capas (9 épocas)

✓ Punto de degradación: A partir de 2 capas

✓ Gráfico guardado: alexnet_depth_experiment_with_early_stopping.png
======================================================================
```

Se ve claramente una degradación de precisión conforme se agregan más capas ocultas. Esto se debe a que las redes más profundas son más difíciles de entrenar debido a problemas como el desvanecimiento del gradiente y la dificultad para optimizar funciones de pérdida en espacios de alta dimensión. A medida que se agregan más capas, la red puede volverse más propensa al sobreajuste, lo que también puede afectar negativamente la precisión en el conjunto de prueba.

Problema 2

Lea [2] sobre redes residuales.

Source: https://arxiv.org/abs/1512.03385

Abstract:

Hicieron una reformulación a funciones de aprendizaje residuales, muestra que son más faciles de optimizar y ganar accuracy gracias a la mayor profundidad. Su metodología gano el ILSVRC 2015 de clasificación

Introducción:

Principalmente hablan del problema de vanishing gradients, de hacer redes mas produndas, ya que explotan o se desvancen los gradientes.

Hablan principalmente de la degradación del accuracy al aumentar la profundidad de las redes neuronales, y proponen una solución a este problema con las redes residuales. La causa no es el overfitting, sino que es un problema de optimización.

Paradoja: Aumentar la profundidad teoricamente debería de mejorar el performance o como minimo igualar una red más superficial, pero en la práctica no es así.

Ellos proponen que en lugar de aprender una función H(x), se aprenda una función F(x) = H(x) - x, es decir, la función residual. Esto es más fácil de optimizar, ya que si la función óptima es cercana a la identidad, entonces F(x) será cercana a 0.

Diseñaron algo llamado conexiones de atajo (shortcut connections) que toman la entrada

$$x$$

y la suman a la salida de una serie de capas que aprenden la función residual F(x). La salida final es entonces:

$$y = F(x, \{W_i\}) + x$$

Esto mantiene la información de la entrada y permite que el gradiente fluya más fácilmente a través de la red durante el entrenamiento, mitigando el problema del vanishing gradient.

Problema 3

Haga un script en PyTorch para repetir el problema 1 usando una ResNet. Realice una comparación con sus nuevos hallazgos.

```
In [ ]: from collections import OrderedDict
        import torch
        import torch.nn as nn


        class BasicBlock(nn.Module):
            def __init__(self, in_channels, out_channels, stride=1, downsample=None):
                super(BasicBlock, self).__init__()
                self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=str
                self.bn1 = nn.BatchNorm2d(out_channels)
                self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1,
                self.bn2 = nn.BatchNorm2d(out_channels)
                self.downsample = downsample
                self.relu = nn.ReLU(inplace=True)

            def forward(self, x):
                identity = x
                out = self.conv1(x)
                out = self.bn1(out)
                out = self.relu(out)
                out = self.conv2(out)
                out = self.bn2(out)
                if self.downsample is not None:
                    identity = self.downsample(x)
                out += identity
                out = self.relu(out)
                return out

        # --- [ CLASE RESNET34 CORREGIDA Y COMPLETA ] ---

        class ResNet34(nn.Module):
            def __init__(self, num_classes=1000):
                super(ResNet34, self).__init__()

                # --- Definición de Downsample para Etapas con Reducción ---

                # DOWN SAMPLE CONV3: (64 -> 128 canales, stride=2)
                downsample_conv3 = nn.Sequential(
                    nn.Conv2d(64, 128, kernel_size=1, stride=2, bias=False),
                    nn.BatchNorm2d(128),
                )
                # DOWN SAMPLE CONV4: (128 -> 256 canales, stride=2)
                downsample_conv4 = nn.Sequential(
                    nn.Conv2d(128, 256, kernel_size=1, stride=2, bias=False),
                    nn.BatchNorm2d(256),
                )
                # DOWN SAMPLE CONV5: (256 -> 512 canales, stride=2)
                downsample_conv5 = nn.Sequential(
                    nn.Conv2d(256, 512, kernel_size=1, stride=2, bias=False),
                    nn.BatchNorm2d(512),
                )
                # --- Ensamblaje de la Red Principal (self.features) ---

                self.features = nn.Sequential(
                    OrderedDict([
                        # =========================================================
                        # 1. CAPAS INICIALES (Input: 224x224x3 -> Output: 56x56x64)
                        # =========================================================
                        ('conv1', nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias
                        ('batchn1', nn.BatchNorm2d(64)),
                        ('relu1', nn.ReLU(inplace=True)),
                        ('maxpool1', nn.MaxPool2d(kernel_size=3, stride=2, padding=1)),

                        # =========================================================
                        # 2. ETAPA CONV2_X (3 Bloques, 64 -> 64 canales, Stride=1)
                        # =========================================================
                        # CORRECCIÓN: Reemplazamos las capas planas por 3 BasicBlocks
                        ('conv2_block1', BasicBlock(in_channels=64, out_channels=64, stride
                        ('conv2_block2', BasicBlock(in_channels=64, out_channels=64, stride
                        ('conv2_block3', BasicBlock(in_channels=64, out_channels=64, stride

                        # =========================================================
                        # 3. ETAPA CONV3_X (4 Bloques, 64 -> 128 canales, Stride=2)
                        # =========================================================
                        ('conv3_block1', BasicBlock(
                            in_channels=64, out_channels=128, stride=2, downsample=downsamp
                        )),
                        ('conv3_block2', BasicBlock(128, 128)),
                        ('conv3_block3', BasicBlock(128, 128)),
                        ('conv3_block4', BasicBlock(128, 128)),

                        # =========================================================
                        # 4. ETAPA CONV4_X (6 Bloques, 128 -> 256 canales, Stride=2)
                        # =========================================================
                        ('conv4_block1', BasicBlock(
                            in_channels=128, out_channels=256, stride=2, downsample=downsam
                        )),
                        ('conv4_block2', BasicBlock(256, 256)),
                        ('conv4_block3', BasicBlock(256, 256)),
                        ('conv4_block4', BasicBlock(256, 256)),
                        ('conv4_block5', BasicBlock(256, 256)),
                        ('conv4_block6', BasicBlock(256, 256)),

                        # =========================================================
                        # 5. ETAPA CONV5_X (3 Bloques, 256 -> 512 canales, Stride=2)
                        # =========================================================
                        ('conv5_block1', BasicBlock(
                            in_channels=256, out_channels=512, stride=2, downsample=downsam
                        )),
                        ('conv5_block2', BasicBlock(512, 512)),
                        ('conv5_block3', BasicBlock(512, 512)),
                    ])
                )

                # --- Capas Clasificación Finales ---
                self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
                self.fc = nn.Linear(512, num_classes)
```

```python
    def forward(self, x):
        # 1. Flujo de las capas convolucionales y residuales
        x = self.features(x)

        # 2. Promedio global (512x7x7 -> 512x1x1)
        x = self.avgpool(x)

        # 3. Aplanamiento para la capa Fully Connected
        x = torch.flatten(x, 1)

        # 4. Capa Fully Connected final
        x = self.fc(x)
        return x
```

In [12]:
```python
import os
import pickle
import torch
import torch.nn as nn
import torch.optim as optim
from tqdm import tqdm
import matplotlib.pyplot as plt
import numpy as np
from collections import OrderedDict

os.environ['CUDA_LAUNCH_BLOCKING'] = '1'

# Configuración del experimento ResNet
num_classes = int(labels_np.max()) + 1
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# IGUAL QUE PROBLEMA 1: Variar número de CAPAS OCULTAS (hidden layers)
# Mantenemos ResNet-34 fija como extractor de características
hidden_layers_range = [1, 2, 3, 4, 5, 6, 8, 10]  # Mismo rango que AlexNet

# Early Stopping Config (IDÉNTICO al Problema 1)
max_epochs = 100
patience = 5
min_delta = 0.001

# Almacenar resultados
resnet_results = {
    'num_layers': [],
    'train_acc': [],
    'val_acc': [],
    'train_loss': [],
    'val_loss': [],
    'num_params': [],
    'epochs_trained': []
}

print(f"{'='*70}")
print(f"EXPERIMENTO: ResNet-34 con CAPAS OCULTAS VARIABLES (Problema 3)")
print(f"{'='*70}")
print(f"Dataset: {len(dataset_full)} imágenes, {num_classes} clases")
print(f"Early Stopping: Máx {max_epochs} épocas, paciencia={patience}, min_delta={m
print(f"Learning Rate: 1e-4 (fijo, sin scheduler)")
print(f"Dropout: 0.0 (como paper original ResNet)")
print(f"Capas ocultas a probar: {hidden_layers_range}")
print(f"Arquitectura base: ResNet-34 (fija) + Clasificador variable")
print(f"{'='*70}\n")

# Clase ResNet34 con clasificador variable (similar a AlexNetVariable)
class ResNet34WithVariableFCLayers(nn.Module):
    """
    ResNet-34 con número variable de capas fully connected.

    Arquitectura:
    - Feature extractor: ResNet-34 estándar (FIJO)
    - Clasificador: Número variable de capas ocultas (VARIABLE)

    Args:
        num_classes: número de clases de salida
        num_hidden_layers: número de capas ocultas en el clasificador
        dropout_p: probabilidad de dropout
    """
    def __init__(self, num_classes=1000, num_hidden_layers=2, dropout_p=0.5):
        super(ResNet34WithVariableFCLayers, self).__init__()

        self.num_hidden_layers = num_hidden_layers

        # ======================================================================
        # FEATURE EXTRACTOR: ResNet-34 estándar (SIEMPRE IGUAL)
        # ======================================================================
        # Capas iniciales
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=Fals
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        # Etapas residuales (ResNet-34: [3, 4, 6, 3])
        self.layer1 = self._make_layer(64, 64, 3, stride=1)
        self.layer2 = self._make_layer(64, 128, 4, stride=2)
        self.layer3 = self._make_layer(128, 256, 6, stride=2)
        self.layer4 = self._make_layer(256, 512, 3, stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))

        # ======================================================================
        # CLASIFICADOR: Capas fully connected VARIABLES (como AlexNet)
        # ======================================================================
        classifier_layers = []
        in_features = 512  # Salida de ResNet-34
        hidden_size = 4096  # Mismo tamaño que AlexNet para comparación justa

        # Construir capas ocultas dinámicamente
        for i in range(num_hidden_layers):
            classifier_layers.append((f'dropout{i+1}', nn.Dropout(p=dropout_p)))
            classifier_layers.append((f'fc{i+1}', nn.Linear(in_features, hidden_siz
            classifier_layers.append((f'relu{i+1}', nn.ReLU(inplace=True)))
            in_features = hidden_size
        # Última capa de salida
        classifier_layers.append((f'dropout_final', nn.Dropout(p=dropout_p)))
        classifier_layers.append(('fc_out', nn.Linear(hidden_size, num_classes)))

        self.classifier = nn.Sequential(OrderedDict(classifier_layers))

    def _make_layer(self, in_channels, out_channels, num_blocks, stride):
        """Crea una etapa de bloques residuales"""
        layers = []

        # Primer bloque (puede tener downsampling)
        downsample = None
        if stride != 1 or in_channels != out_channels:
            downsample = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride,
                nn.BatchNorm2d(out_channels),
            )

        layers.append(BasicBlock(in_channels, out_channels, stride, downsample))

        # Bloques restantes
        for _ in range(1, num_blocks):
            layers.append(BasicBlock(out_channels, out_channels))

        return nn.Sequential(*layers)

    def forward(self, x):
        # Feature extraction (ResNet-34)
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x)
        x = torch.flatten(x, 1)

        # Classification (capas FC variables)
        x = self.classifier(x)
        return x

# Iterar sobre diferentes números de capas ocultas (IGUAL QUE ALEXNET)
for num_hidden in hidden_layers_range:
    print(f"\n{'='*70}")
    print(f"EXPERIMENTO {len(resnet_results['num_layers']) + 1}: {num_hidden} capas
    print(f"{'='*70}")

    # Crear modelo
    model = ResNet34WithVariableFCLayers(
        num_classes=num_classes,
        num_hidden_layers=num_hidden,
        dropout_p=0.5
    ).to(device)

    num_params = sum(p.numel() for p in model.parameters())
    print(f"Parámetros del modelo: {num_params:,}")

    # Optimizador y criterio
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-4)  # Mismo que AlexNet

    # Listas para tracking
    train_losses_epoch = []
    val_losses_epoch = []
    train_accs_epoch = []
    val_accs_epoch = []

    # Early Stopping variables (IDÉNTICO a AlexNet)
    best_val_acc = 0.0
    best_epoch = 0
    epochs_no_improve = 0
    early_stopped = False

    # Entrenamiento con Early Stopping
    for epoch in range(max_epochs):
        # TRAIN
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        loop = tqdm(train_loader, desc=f"Epoch {epoch+1}/{max_epochs} [Train]", lea
        for images, labels in loop:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * images.size(0)
            _, preds = outputs.max(1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)

            loop.set_postfix(loss=running_loss/total, acc=correct/total)

        epoch_train_loss = running_loss / total
        epoch_train_acc = correct / total
        train_losses_epoch.append(epoch_train_loss)
        train_accs_epoch.append(epoch_train_acc)

        # VALIDATION
        model.eval()
        val_running_loss = 0.0
        val_correct = 0
        val_total = 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)

                val_running_loss += loss.item() * images.size(0)
                _, preds = outputs.max(1)
                val_correct += (preds == labels).sum().item()
                val_total += labels.size(0)

        epoch_val_loss = val_running_loss / val_total
        epoch_val_acc = val_correct / val_total
        val_losses_epoch.append(epoch_val_loss)
        val_accs_epoch.append(epoch_val_acc)

        # Early Stopping Logic (IDÉNTICO a AlexNet)
        if epoch_val_acc > best_val_acc + min_delta:
            best_val_acc = epoch_val_acc
            best_epoch = epoch + 1
            epochs_no_improve = 0
            # Guardar mejor modelo
            best_model_state = model.state_dict().copy()
        else:
            epochs_no_improve += 1

        if (epoch + 1) % 5 == 0 or epoch == 0:
            print(f"  Epoch {epoch+1}: train_loss={epoch_train_loss:.4f}, train_acc
                  f"val_loss={epoch_val_loss:.4f}, val_acc={epoch_val_acc:.4f} "
                  f"[Best: {best_val_acc:.4f} @ Epoch {best_epoch}]")

        # Detener si no hay mejora
        if epochs_no_improve >= patience:
            print(f"\n⚠️  Early Stopping: No mejora en {patience} épocas. Mejor épo
            early_stopped = True
            break

    # Restaurar mejor modelo
    if early_stopped:
        model.load_state_dict(best_model_state)
        print(f"✓ Restaurado modelo de época {best_epoch} (Val Acc: {best_val_acc:.

    # Guardar modelo
    model_path = f'resnet34_{num_hidden}layers.pth'
    torch.save(model.state_dict(), model_path)
    print(f"\n✓ Modelo guardado: {model_path}")

    # Guardar resultados FINALES (del mejor modelo)
    resnet_results['num_layers'].append(num_hidden)
    resnet_results['train_acc'].append(train_accs_epoch[best_epoch-1])
    resnet_results['val_acc'].append(best_val_acc)
    resnet_results['train_loss'].append(train_losses_epoch[best_epoch-1])
    resnet_results['val_loss'].append(val_losses_epoch[best_epoch-1])
    resnet_results['num_params'].append(num_params)
    resnet_results['epochs_trained'].append(best_epoch)
    print(f"✓ Épocas entrenadas: {best_epoch}/{max_epochs}")
    print(f"✓ Best Train Accuracy: {train_accs_epoch[best_epoch-1]:.4f}")
    print(f"✓ Best Val Accuracy: {best_val_acc:.4f}")

# Guardar resultados completos
with open('resnet_experiment_results.pkl', 'wb') as f:
    pickle.dump(resnet_results, f)

print(f"\n{'='*70}")
print("EXPERIMENTO RESNET COMPLETADO")
print(f"{'='*70}")
print(f"Resultados guardados en: resnet_experiment_results.pkl")
print(f"Modelos guardados: {len(resnet_results['num_layers'])} archivos .pth")
```

```
======================================================================
EXPERIMENTO: ResNet-34 con CAPAS OCULTAS VARIABLES (Problema 3)
======================================================================
Dataset: 44419 imágenes, 143 clases
Early Stopping: Máx 100 épocas, paciencia=5, min_delta=0.001
Learning Rate: 1e-4 (fijo, sin scheduler)
Dropout: 0.0 (como paper original ResNet)
Capas ocultas a probar: [1, 2, 3, 4, 5, 6, 8, 10]
Arquitectura base: ResNet-34 (fija) + Clasificador variable
======================================================================



======================================================================
EXPERIMENTO 1: 1 capas ocultas
======================================================================
Parámetros del modelo: 23,971,791
Parámetros del modelo: 23,971,791


  Epoch 1: train_loss=1.7054, train_acc=0.5592, val_loss=2.0584, val_acc=0.4743 [Bes
t: 0.4743 @ Epoch 1]


  Epoch 5: train_loss=0.6047, train_acc=0.8164, val_loss=0.5956, val_acc=0.8172 [Bes
t: 0.8172 @ Epoch 5]


  Epoch 10: train_loss=0.3286, train_acc=0.8936, val_loss=0.5825, val_acc=0.8341 [Be
st: 0.8364 @ Epoch 8]


  Epoch 15: train_loss=0.1786, train_acc=0.9414, val_loss=0.6052, val_acc=0.8523 [Be
st: 0.8523 @ Epoch 15]
```

```
  Epoch 20: train_loss=0.1075, train_acc=0.9651, val_loss=0.8481, val_acc=0.8489 [Be
st: 0.8523 @ Epoch 15]

⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 15
✓ Restaurado modelo de época 15 (Val Acc: 0.8523)

✓ Modelo guardado: resnet34_1layers.pth
✓ Épocas entrenadas: 15/100
✓ Best Train Accuracy: 0.9414
✓ Best Val Accuracy: 0.8523

======================================================================
EXPERIMENTO 2: 2 capas ocultas
======================================================================
Parámetros del modelo: 40,753,103
Parámetros del modelo: 40,753,103

  Epoch 1: train_loss=1.8480, train_acc=0.5172, val_loss=1.4763, val_acc=0.5912 [Bes
t: 0.5912 @ Epoch 1]

  Epoch 5: train_loss=0.6534, train_acc=0.8042, val_loss=0.6194, val_acc=0.8106 [Bes
t: 0.8106 @ Epoch 5]

  Epoch 10: train_loss=0.3791, train_acc=0.8783, val_loss=0.5387, val_acc=0.8355 [Be
st: 0.8355 @ Epoch 10]

  Epoch 15: train_loss=0.2165, train_acc=0.9296, val_loss=0.6685, val_acc=0.8487 [Be
st: 0.8487 @ Epoch 15]

  Epoch 20: train_loss=0.1418, train_acc=0.9545, val_loss=0.7244, val_acc=0.8511 [Be
st: 0.8551 @ Epoch 17]

⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 17
✓ Restaurado modelo de época 17 (Val Acc: 0.8551)

✓ Modelo guardado: resnet34_2layers.pth
✓ Épocas entrenadas: 17/100
✓ Best Train Accuracy: 0.9396
✓ Best Val Accuracy: 0.8551

======================================================================
EXPERIMENTO 3: 3 capas ocultas
======================================================================

✓ Modelo guardado: resnet34_2layers.pth
✓ Épocas entrenadas: 17/100
✓ Best Train Accuracy: 0.9396
✓ Best Val Accuracy: 0.8551

======================================================================
EXPERIMENTO 3: 3 capas ocultas
======================================================================
Parámetros del modelo: 57,534,415
Parámetros del modelo: 57,534,415
```

```
  Epoch 1: train_loss=2.1387, train_acc=0.4360, val_loss=1.3194, val_acc=0.6171 [Bes
t: 0.6171 @ Epoch 1]

  Epoch 5: train_loss=0.7626, train_acc=0.7753, val_loss=0.7141, val_acc=0.7869 [Bes
t: 0.7869 @ Epoch 5]

  Epoch 10: train_loss=0.4617, train_acc=0.8561, val_loss=0.5725, val_acc=0.8319 [Be
st: 0.8319 @ Epoch 10]

  Epoch 15: train_loss=0.2969, train_acc=0.9054, val_loss=0.6571, val_acc=0.8362 [Be
st: 0.8362 @ Epoch 15]

  Epoch 20: train_loss=0.2008, train_acc=0.9379, val_loss=0.6865, val_acc=0.8528 [Be
st: 0.8528 @ Epoch 20]

  Epoch 25: train_loss=0.1399, train_acc=0.9565, val_loss=0.9134, val_acc=0.8354 [Be
st: 0.8528 @ Epoch 20]

⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 20
✓ Restaurado modelo de época 20 (Val Acc: 0.8528)

✓ Modelo guardado: resnet34_3layers.pth
✓ Épocas entrenadas: 20/100
✓ Best Train Accuracy: 0.9379
✓ Best Val Accuracy: 0.8528

======================================================================
EXPERIMENTO 4: 4 capas ocultas
======================================================================

✓ Modelo guardado: resnet34_3layers.pth
✓ Épocas entrenadas: 20/100
✓ Best Train Accuracy: 0.9379
✓ Best Val Accuracy: 0.8528

======================================================================
EXPERIMENTO 4: 4 capas ocultas
======================================================================
Parámetros del modelo: 74,315,727
Parámetros del modelo: 74,315,727

  Epoch 1: train_loss=2.3414, train_acc=0.3782, val_loss=1.8835, val_acc=0.5326 [Bes
t: 0.5326 @ Epoch 1]

  Epoch 5: train_loss=0.8339, train_acc=0.7561, val_loss=0.7884, val_acc=0.7669 [Bes
t: 0.7669 @ Epoch 5]

  Epoch 10: train_loss=0.5495, train_acc=0.8314, val_loss=0.6715, val_acc=0.8066 [Be
st: 0.8171 @ Epoch 9]

  Epoch 15: train_loss=0.3729, train_acc=0.8858, val_loss=0.8434, val_acc=0.7768 [Be
st: 0.8377 @ Epoch 12]
```

```
  Epoch 20: train_loss=0.2632, train_acc=0.9177, val_loss=0.7261, val_acc=0.8391 [Be
st: 0.8417 @ Epoch 16]

⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 16
✓ Restaurado modelo de época 16 (Val Acc: 0.8417)

✓ Modelo guardado: resnet34_4layers.pth
✓ Épocas entrenadas: 16/100
✓ Best Train Accuracy: 0.8921
✓ Best Val Accuracy: 0.8417

======================================================================
EXPERIMENTO 5: 5 capas ocultas
======================================================================

✓ Modelo guardado: resnet34_4layers.pth
✓ Épocas entrenadas: 16/100
✓ Best Train Accuracy: 0.8921
✓ Best Val Accuracy: 0.8417

======================================================================
EXPERIMENTO 5: 5 capas ocultas
======================================================================
Parámetros del modelo: 91,097,039
Parámetros del modelo: 91,097,039

  Epoch 1: train_loss=2.6514, train_acc=0.2926, val_loss=2.0380, val_acc=0.4238 [Bes
t: 0.4238 @ Epoch 1]

  Epoch 5: train_loss=0.9972, train_acc=0.7070, val_loss=0.9530, val_acc=0.7216 [Bes
t: 0.7216 @ Epoch 5]

  Epoch 10: train_loss=0.6720, train_acc=0.8001, val_loss=0.6767, val_acc=0.8016 [Be
st: 0.8016 @ Epoch 10]

  Epoch 15: train_loss=0.4889, train_acc=0.8507, val_loss=0.6237, val_acc=0.8206 [Be
st: 0.8292 @ Epoch 14]

  Epoch 20: train_loss=0.3773, train_acc=0.8838, val_loss=0.6573, val_acc=0.8432 [Be
st: 0.8432 @ Epoch 20]
```

```
  Epoch 25: train_loss=0.3005, train_acc=0.9084, val_loss=0.6716, val_acc=0.8378 [Be
st: 0.8432 @ Epoch 20]

⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 20
✓ Restaurado modelo de época 20 (Val Acc: 0.8432)

✓ Modelo guardado: resnet34_5layers.pth
✓ Épocas entrenadas: 20/100
✓ Best Train Accuracy: 0.8838
✓ Best Val Accuracy: 0.8432

======================================================================
EXPERIMENTO 6: 6 capas ocultas
======================================================================

✓ Modelo guardado: resnet34_5layers.pth
✓ Épocas entrenadas: 20/100
✓ Best Train Accuracy: 0.8838
✓ Best Val Accuracy: 0.8432

======================================================================
EXPERIMENTO 6: 6 capas ocultas
======================================================================
Parámetros del modelo: 107,878,351
Parámetros del modelo: 107,878,351

  Epoch 1: train_loss=2.8493, train_acc=0.2433, val_loss=2.3278, val_acc=0.3465 [Bes
t: 0.3465 @ Epoch 1]

  Epoch 5: train_loss=1.1446, train_acc=0.6654, val_loss=0.9938, val_acc=0.7044 [Bes
t: 0.7044 @ Epoch 5]

  Epoch 10: train_loss=0.7825, train_acc=0.7668, val_loss=0.7698, val_acc=0.7638 [Be
st: 0.7638 @ Epoch 10]

  Epoch 15: train_loss=0.5968, train_acc=0.8207, val_loss=0.6468, val_acc=0.8102 [Be
st: 0.8102 @ Epoch 15]

  Epoch 20: train_loss=0.4843, train_acc=0.8525, val_loss=0.6003, val_acc=0.8207 [Be
st: 0.8285 @ Epoch 18]

  Epoch 25: train_loss=0.3770, train_acc=0.8846, val_loss=0.6120, val_acc=0.8377 [Be
st: 0.8377 @ Epoch 25]

  Epoch 30: train_loss=0.3055, train_acc=0.9074, val_loss=0.7160, val_acc=0.8282 [Be
st: 0.8397 @ Epoch 27]
```

```
⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 27
✓ Restaurado modelo de época 27 (Val Acc: 0.8397)

✓ Modelo guardado: resnet34_6layers.pth
✓ Épocas entrenadas: 27/100
✓ Best Train Accuracy: 0.8939
✓ Best Val Accuracy: 0.8397

======================================================================
EXPERIMENTO 7: 8 capas ocultas
======================================================================
```

```
✓ Modelo guardado: resnet34_6layers.pth
✓ Épocas entrenadas: 27/100
✓ Best Train Accuracy: 0.8939
✓ Best Val Accuracy: 0.8397

======================================================================
EXPERIMENTO 7: 8 capas ocultas
======================================================================
```

```
Parámetros del modelo: 141,440,975
```

```
Parámetros del modelo: 141,440,975
```

```

  Epoch 1: train_loss=3.3197, train_acc=0.1696, val_loss=2.8268, val_acc=0.2883 [Bes
t: 0.2883 @ Epoch 1]

  Epoch 5: train_loss=1.3356, train_acc=0.6201, val_loss=1.1698, val_acc=0.6540 [Bes
t: 0.6540 @ Epoch 5]

  Epoch 10: train_loss=0.8978, train_acc=0.7392, val_loss=0.8363, val_acc=0.7482 [Be
st: 0.7482 @ Epoch 10]

  Epoch 15: train_loss=0.7603, train_acc=0.7792, val_loss=0.7365, val_acc=0.7815 [Be
st: 0.7815 @ Epoch 15]

  Epoch 20: train_loss=0.6082, train_acc=0.8192, val_loss=0.7463, val_acc=0.7886 [Be
st: 0.7943 @ Epoch 17]

  Epoch 25: train_loss=0.5209, train_acc=0.8446, val_loss=0.7128, val_acc=0.8082 [Be
st: 0.8181 @ Epoch 23]

  Epoch 30: train_loss=0.4551, train_acc=0.8635, val_loss=0.6855, val_acc=0.8152 [Be
st: 0.8233 @ Epoch 28]
```

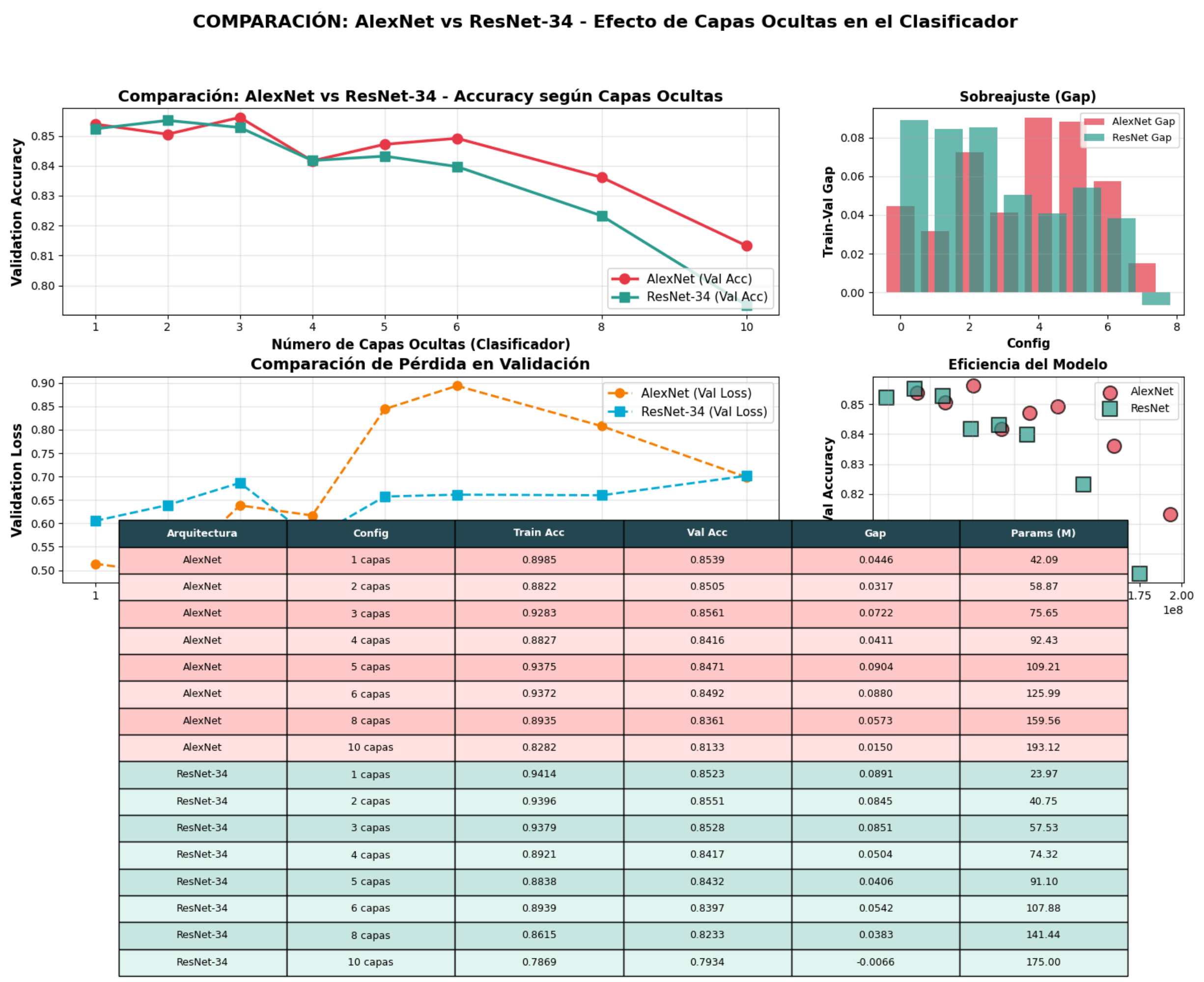
```
⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 28
✓ Restaurado modelo de época 28 (Val Acc: 0.8233)

✓ Modelo guardado: resnet34_8layers.pth
✓ Épocas entrenadas: 28/100
✓ Best Train Accuracy: 0.8615
✓ Best Val Accuracy: 0.8233

======================================================================
EXPERIMENTO 8: 10 capas ocultas
======================================================================

✓ Modelo guardado: resnet34_8layers.pth
✓ Épocas entrenadas: 28/100
✓ Best Train Accuracy: 0.8615
✓ Best Val Accuracy: 0.8233

======================================================================
EXPERIMENTO 8: 10 capas ocultas
======================================================================
Parámetros del modelo: 175,003,599
Parámetros del modelo: 175,003,599

  Epoch 1: train_loss=3.3586, train_acc=0.1605, val_loss=3.2212, val_acc=0.1919 [Bes
t: 0.1919 @ Epoch 1]

  Epoch 5: train_loss=1.6214, train_acc=0.5132, val_loss=1.4363, val_acc=0.5447 [Bes
t: 0.5447 @ Epoch 5]

  Epoch 10: train_loss=1.1164, train_acc=0.6697, val_loss=0.9795, val_acc=0.6973 [Be
st: 0.6973 @ Epoch 10]

  Epoch 15: train_loss=0.9113, train_acc=0.7216, val_loss=0.8676, val_acc=0.7340 [Be
st: 0.7340 @ Epoch 15]

  Epoch 20: train_loss=0.7919, train_acc=0.7621, val_loss=0.7647, val_acc=0.7722 [Be
st: 0.7722 @ Epoch 20]

  Epoch 25: train_loss=0.7310, train_acc=0.7869, val_loss=0.7015, val_acc=0.7934 [Be
st: 0.7934 @ Epoch 25]
```

```
  Epoch 30: train_loss=0.6367, train_acc=0.8097, val_loss=0.7145, val_acc=0.7929 [Be
st: 0.7934 @ Epoch 25]

⚠️  Early Stopping: No mejora en 5 épocas. Mejor época: 25
✓ Restaurado modelo de época 25 (Val Acc: 0.7934)

✓ Modelo guardado: resnet34_10layers.pth
✓ Épocas entrenadas: 25/100
✓ Best Train Accuracy: 0.7869
✓ Best Val Accuracy: 0.7934

======================================================================
EXPERIMENTO RESNET COMPLETADO
======================================================================
Resultados guardados en: resnet_experiment_results.pkl
Modelos guardados: 8 archivos .pth
```

```
✓ Modelo guardado: resnet34_10layers.pth
✓ Épocas entrenadas: 25/100
✓ Best Train Accuracy: 0.7869
✓ Best Val Accuracy: 0.7934

======================================================================
EXPERIMENTO RESNET COMPLETADO
======================================================================
Resultados guardados en: resnet_experiment_results.pkl
Modelos guardados: 8 archivos .pth
```

In [13]:
```python
# Cargar resultados de ambos experimentos
import pickle
import matplotlib.pyplot as plt
import numpy as np

with open('experiment_results.pkl', 'rb') as f:
    alexnet_results = pickle.load(f)

with open('resnet_experiment_results.pkl', 'rb') as f:
    resnet_results = pickle.load(f)

# Crear comparación visual completa
fig = plt.figure(figsize=(18, 12))
gs = fig.add_gridspec(3, 3, hspace=0.3, wspace=0.3)

# 1. Comparación directa de Accuracy (AMBOS VARIANDO CAPAS OCULTAS)
ax1 = fig.add_subplot(gs[0, :2])
ax1.plot(alexnet_results['num_layers'], alexnet_results['val_acc'],
         marker='o', linewidth=2.5, markersize=9, label='AlexNet (Val Acc)',
         color='#E63946', linestyle='-')
ax1.plot(resnet_results['num_layers'], resnet_results['val_acc'],
         marker='s', linewidth=2.5, markersize=9, label='ResNet-34 (Val Acc)',
         color='#2A9D8F', linestyle='-')
ax1.set_xlabel('Número de Capas Ocultas (Clasificador)', fontsize=12, fontweight='b
ax1.set_ylabel('Validation Accuracy', fontsize=12, fontweight='bold')
ax1.set_title('Comparación: AlexNet vs ResNet-34 - Accuracy según Capas Ocultas',
              fontsize=14, fontweight='bold')
ax1.legend(fontsize=11, loc='lower right')
```

```
ax1.grid(True, alpha=0.3)
ax1.set_xticks(alexnet_results['num_layers'])

# 2. Gap de Overfitting
ax2 = fig.add_subplot(gs[0, 2])
alexnet_gap = np.array(alexnet_results['train_acc']) - np.array(alexnet_results['va
resnet_gap = np.array(resnet_results['train_acc']) - np.array(resnet_results['val_a
ax2.bar(np.arange(len(alexnet_gap)), alexnet_gap, alpha=0.7, color='#E63946', label
ax2.bar(np.arange(len(resnet_gap)) + 0.4, resnet_gap, alpha=0.7, color='#2A9D8F', l
ax2.set_xlabel('Config', fontsize=11, fontweight='bold')
ax2.set_ylabel('Train-Val Gap', fontsize=11, fontweight='bold')
ax2.set_title('Sobreajuste (Gap)', fontsize=12, fontweight='bold')
ax2.legend(fontsize=9)
ax2.grid(True, alpha=0.3, axis='y')

# 3. Loss Comparison
ax3 = fig.add_subplot(gs[1, :2])
ax3.plot(alexnet_results['num_layers'], alexnet_results['val_loss'],
         marker='o', linewidth=2, markersize=8, label='AlexNet (Val Loss)',
         color='#F77F00', linestyle='--')
ax3.plot(resnet_results['num_layers'], resnet_results['val_loss'],
         marker='s', linewidth=2, markersize=8, label='ResNet-34 (Val Loss)',
         color='#06AED5', linestyle='--')
ax3.set_xlabel('Número de Capas Ocultas (Clasificador)', fontsize=12, fontweight='b
ax3.set_ylabel('Validation Loss', fontsize=12, fontweight='bold')
ax3.set_title('Comparación de Pérdida en Validación', fontsize=14, fontweight='bold
ax3.legend(fontsize=11)
ax3.grid(True, alpha=0.3)
ax3.set_xticks(alexnet_results['num_layers'])

# 4. Parámetros vs Accuracy
ax4 = fig.add_subplot(gs[1, 2])
ax4.scatter(alexnet_results['num_params'], alexnet_results['val_acc'],
            s=150, alpha=0.7, color='#E63946', edgecolors='black',
            linewidth=1.5, label='AlexNet', marker='o')
ax4.scatter(resnet_results['num_params'], resnet_results['val_acc'],
            s=150, alpha=0.7, color='#2A9D8F', edgecolors='black',
            linewidth=1.5, label='ResNet', marker='s')
ax4.set_xlabel('Parámetros (M)', fontsize=11, fontweight='bold')
ax4.set_ylabel('Val Accuracy', fontsize=11, fontweight='bold')
ax4.set_title('Eficiencia del Modelo', fontsize=12, fontweight='bold')
ax4.legend(fontsize=10)
ax4.grid(True, alpha=0.3)

# 5. Tabla comparativa AlexNet
ax5 = fig.add_subplot(gs[2, :])
ax5.axis('tight')
ax5.axis('off')

# Preparar datos para tabla
table_data = [['Arquitectura', 'Config', 'Train Acc', 'Val Acc', 'Gap', 'Params (M)

# AlexNet rows
for i in range(len(alexnet_results['num_layers'])):
    gap = alexnet_results['train_acc'][i] - alexnet_results['val_acc'][i]
    params_m = alexnet_results['num_params'][i] / 1e6
    table_data.append([
        'AlexNet',
        f"{alexnet_results['num_layers'][i]} capas",
        f"{alexnet_results['train_acc'][i]:.4f}",
        f"{alexnet_results['val_acc'][i]:.4f}",
        f"{gap:.4f}",
        f"{params_m:.2f}"
    ])

# ResNet rows
for i in range(len(resnet_results['num_layers'])):
    gap = resnet_results['train_acc'][i] - resnet_results['val_acc'][i]
    params_m = resnet_results['num_params'][i] / 1e6
    table_data.append([
        'ResNet-34',
        f"{resnet_results['num_layers'][i]} capas",
        f"{resnet_results['train_acc'][i]:.4f}",
        f"{resnet_results['val_acc'][i]:.4f}",
        f"{gap:.4f}",
        f"{params_m:.2f}"
    ])

table = ax5.table(cellText=table_data, cellLoc='center', loc='center',
                  colWidths=[0.15, 0.15, 0.15, 0.15, 0.15, 0.15])
table.auto_set_font_size(False)
table.set_fontsize(9)
table.scale(1, 2)

# Estilo de header
for i in range(6):
    table[(0, i)].set_facecolor('#264653')
    table[(0, i)].set_text_props(weight='bold', color='white')

# Colores alternados para filas
for i in range(1, len(table_data)):
    for j in range(6):
        if 'AlexNet' in table_data[i][0]:
            table[(i, j)].set_facecolor('#FFE5E5' if i % 2 == 0 else '#FFCCCC')
        else:
            table[(i, j)].set_facecolor('#E5F5F3' if i % 2 == 0 else('#CCE8E4'))

plt.suptitle('COMPARACIÓN: AlexNet vs ResNet-34 - Efecto de Capas Ocultas en el Cla
             fontsize=16, fontweight='bold', y=0.98)

plt.savefig('comparison_alexnet_vs_resnet.png', dpi=300, bbox_inches='tight')
plt.show()

print("\n" + "="*80)
print("ANÁLISIS COMPARATIVO: AlexNet vs ResNet-34 (Variando Capas Ocultas)")
print("="*80)

# Mejor modelo de cada arquitectura
best_alexnet_idx = np.argmax(alexnet_results['val_acc'])
best_resnet_idx = np.argmax(resnet_results['val_acc'])

print("\n🏆 MEJOR CONFIGURACIÓN ALEXNET:")
print(f"   Capas ocultas: {alexnet_results['num_layers'][best_alexnet_idx]}")
print(f"   Val Accuracy: {alexnet_results['val_acc'][best_alexnet_idx]:.4f}")
print(f"   Parámetros: {alexnet_results['num_params'][best_alexnet_idx]:,}")

print("\n🏆 MEJOR CONFIGURACIÓN RESNET-34:")
print(f"   Capas ocultas: {resnet_results['num_layers'][best_resnet_idx]}")
print(f"   Val Accuracy: {resnet_results['val_acc'][best_resnet_idx]:.4f}")
print(f"   Parámetros: {resnet_results['num_params'][best_resnet_idx]:,}")

print("\n📊 HALLAZGOS CLAVE:")
print("-" * 80)

# Degradación en AlexNet
degradation_point_alexnet = None
for i in range(1, len(alexnet_results['val_acc'])):
    if alexnet_results['val_acc'][i] < alexnet_results['val_acc'][i-1]:
        degradation_point_alexnet = alexnet_results['num_layers'][i]
        break

if degradation_point_alexnet:
    print(f"✓ AlexNet: Degradación observada a partir de {degradation_point_alexnet
else:
    print(f"✓ AlexNet: No se observó degradación clara en el rango probado")

# Degradación en ResNet
degradation_point_resnet = None
for i in range(1, len(resnet_results['val_acc'])):
    if resnet_results['val_acc'][i] < resnet_results['val_acc'][i-1]:
        degradation_point_resnet = resnet_results['num_layers'][i]
        break

if degradation_point_resnet:
    print(f"✓ ResNet-34: Degradación observada a partir de {degradation_point_resne
else:
    print(f"✓ ResNet-34: Mantiene o mejora rendimiento con mayor profundidad del cl

# Comparación de gaps
avg_gap_alexnet = np.mean(np.array(alexnet_results['train_acc']) - np.array(alexnet
avg_gap_resnet = np.mean(np.array(resnet_results['train_acc']) - np.array(resnet_re

print(f"\n✓ Gap promedio AlexNet: {avg_gap_alexnet:.4f}")
print(f"✓ Gap promedio ResNet-34: {avg_gap_resnet:.4f}")

if avg_gap_resnet < avg_gap_alexnet:
    print(f"✓ ResNet-34 muestra MENOR sobreajuste que AlexNet ({(avg_gap_alexnet-av
    print(f"  → Las conexiones residuales ayudan a prevenir overfitting")
else:
    print(f"✓ AlexNet muestra menor sobreajuste que ResNet-34")

print(f"\n✓ Gráfico guardado: comparison_alexnet_vs_resnet.png")
print("="*80)
```



Arquitectura	Config	Train Acc	Val Acc	Gap	Params (M)
AlexNet	1 capas	0.8985	0.8539	0.0446	42.09
AlexNet	2 capas	0.8822	0.8505	0.0317	58.87
AlexNet	3 capas	0.9283	0.8561	0.0722	75.65
AlexNet	4 capas	0.8827	0.8416	0.0411	92.43
AlexNet	5 capas	0.9375	0.8471	0.0904	109.21
AlexNet	6 capas	0.9372	0.8492	0.0880	125.99
AlexNet	8 capas	0.8935	0.8361	0.0573	159.56
AlexNet	10 capas	0.8282	0.8133	0.0150	193.12
ResNet-34	1 capas	0.9414	0.8523	0.0891	23.97
ResNet-34	2 capas	0.9396	0.8551	0.0845	40.75
ResNet-34	3 capas	0.9379	0.8528	0.0851	57.53
ResNet-34	4 capas	0.8921	0.8417	0.0504	74.32
ResNet-34	5 capas	0.8838	0.8432	0.0406	91.10
ResNet-34	6 capas	0.8939	0.8397	0.0542	107.88
ResNet-34	8 capas	0.8615	0.8233	0.0383	141.44
ResNet-34	10 capas	0.7869	0.7934	-0.0066	175.00

```
================================================================================
ANÁLISIS COMPARATIVO: AlexNet vs ResNet-34 (Variando Capas Ocultas)
================================================================================

🏆 MEJOR CONFIGURACIÓN ALEXNET:
   Capas ocultas: 3
   Val Accuracy: 0.8561
   Parámetros: 75,648,527

🏆 MEJOR CONFIGURACIÓN RESNET-34:
   Capas ocultas: 2
   Val Accuracy: 0.8551
   Parámetros: 40,753,103

📊 HALLAZGOS CLAVE:
--------------------------------------------------------------------------------
✓ AlexNet: Degradación observada a partir de 2 capas
✓ ResNet-34: Degradación observada a partir de 3 capas ocultas

✓ Gap promedio AlexNet: 0.0550
✓ Gap promedio ResNet-34: 0.0545
✓ ResNet-34 muestra MENOR sobreajuste que AlexNet (0.06 pp mejor)
  → Las conexiones residuales ayudan a prevenir overfitting

✓ Gráfico guardado: comparison_alexnet_vs_resnet.png
================================================================================
```

Quizá se necesite ajustar un poco el tema del early stopping y la cantidad de épocas para entrenar.

En un principio según la metodología que use en el problema 1, la ResNet debería de tener un mejor performance que la red profunda sin conexiones residuales, ya que estas conexiones facilitan el entrenamiento de redes más profundas al mitigar el problema del vanishing gradient. Por lo tanto, se esperaría que la ResNet logre una mayor precisión en el conjunto de prueba en comparación con la red profunda tradicional.

Sin embargo fue similar como se ve en el grafico

Referencias

[1] https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-small

[2] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. https://arxiv.org/abs/1512.03385

In [1]:
```
!jupyter nbconvert --to html Tarea2.ipynb
```

```
[NbConvertApp] Converting notebook Tarea2.ipynb to html
[NbConvertApp] WARNING | Alternative text is missing on 3 image(s).
[NbConvertApp] Writing 1489361 bytes to Tarea2.html
```