HAFTA 2 • CUMARTESİ

Evrişimli Sinir Ağları & Transfer Learning

Conv2D & Feature Maps → Pooling → Data Augmentation → ResNet Blokları → Fine-Tuning

09:00 — 17:00 · 7 Saat · 4 Python Dosyası · 700+ Satır Kod · Veri: CIFAR-10

Günün Planı — Hafta 2 Cumartesi

09:00–09:30

Geçen Hafta Özeti & CNN'e Giriş

09:30–10:45

Conv2D, Feature Map, Pooling Lab

10:45–11:00

Kısa Mola

11:00–12:00

Data Augmentation Stratejileri

12:00–13:00

İleri CNN: ResNet, DepthwiseConv

13:00–14:00

Öğle Arası

14:00–15:30

Transfer Learning & Fine-Tuning

15:30–16:30

Model Analizi & Grad-CAM

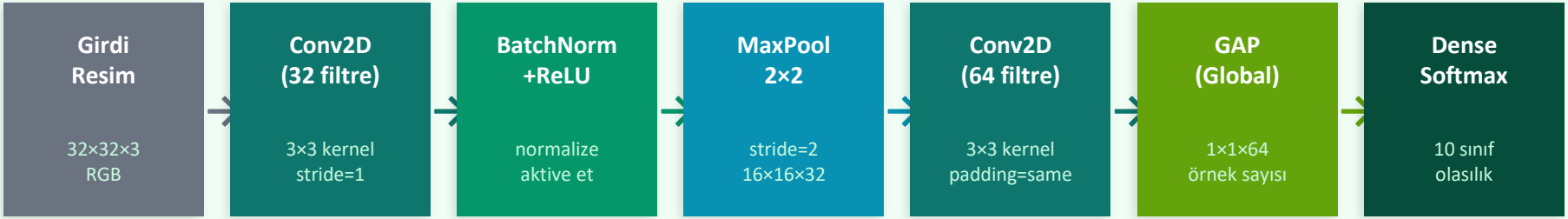
16:30–17:00

Mini Proje & Kapanış

Gün Sonu: CIFAR-10'da %85+ accuracy + Pre-trained model + Grad-CAM görselleştirme

CNN Temel Mimarisi — Katmandan Katmana

Conv2D → BatchNorm → Activation → Pooling → Dense



Filtre (Kernel)

Ağırlık matrisi (ör. 3×3). Girdi üzerinde kaydırılarak özellik haritası üretir. Ağ ağırlıkları öğrenilir.

Feature Map

Bir filtrenin girdiyle konvolüsyon sonucu. Her filtre farklı bir özelliği yakalar (kenar, doku, şekil).

Padding='same'

Çıktı boyutunu girdiyle aynı tutar. 'valid'=küçülür. 'same'=aynı kalır. Kenar bilgisi korunur.

Global Avg Pooling

Her feature map'teki tüm değerlerin ortalaması. Flatten yerine parametre sayısını dramatik düşürür.

Conv2D — Derinlemesine Parametre Analizi

filters · kernel_size · strides · padding · dilation_rate · groups

conv2d_detay.py

```
# ── TEMEL CONV2D KULLANIMI ──────────────────────────
x = layers.Conv2D(
    filters=64,            # Öğrenilecek filtre sayısı
    kernel_size=(3,3),     # Filtre boyutu (veya sadece 3)
    strides=(1,1),         # Kaydırma adımı
    padding='same',        # 'valid' veya 'same'
    activation=None,       # BN önce None tercih edilir
    use_bias=False,        # BatchNorm varsa bias gereksiz
    kernel_initializer='he_normal',
    kernel_regularizer=regularizers.l2(1e-4),
    name='conv_block1'
)(x)

# ── DEPTHWISE SEPARABLE CONV (verimli) ──────────────
x = layers.DepthwiseConv2D(
    kernel_size=3, padding='same', use_bias=False)(x)
x = layers.Conv2D(64, 1, use_bias=False)(x)   # pointwise

# ── DİLATED CONV (büyük reseptif alan) ──────────────
x = layers.Conv2D(
    64, 3, padding='same',
    dilation_rate=2       # 2=etkin 5×5, 4=etkin 9×9 k=k+(k-1)(d-1)
)(x)

#   PARAMETRE SAYISI
# params = (3×3×32 + 1) × 64 = 18.496
# DSConv aynı yapı:
```

filters=64

64 farklı özelliği yakalar. Derin katmanlarda artırılır (32→64→128→256). Her filtre bağımsız öğrenir.

kernel_size=3

3×3 en yaygın. Büyük kernel=geniş görüş alanı ama daha fazla parametre. 1×1 boyut dönüşümü için.

strides=2

Stride=2 → feature map yarı boyuta düşer. MaxPool yerine stride=2 Conv kullanmak modern tercih.

padding='same'

Çıktı = Girdi boyutu. Kenar pikseller tekrarlanır. UNet, segmentasyon için kritik.

dilation_rate=2

Filtre boşluklu uygulanır. 3×3 filtre → 5×5 alana bakar. Segmentasyon, ses işlemede yaygın.

DSConv = DepthwiseConv2D + Conv2D(1×1) → %87 daha az parametre, benzer doğruluk (MobileNet prensibi)

Pooling & Normalization — Karşılaştırmalı Analiz

MaxPool · AvgPool · GlobalAvgPool · BatchNorm · LayerNorm · GroupNorm

MaxPooling2D(2×2)

`max(2×2 bölge)`

✓ Keskin özellik; kenar/doku iyi algılar

⚠ Konumsal bilgi kaybolabilir

AveragePooling2D(2×2)

`mean(2×2 bölge)`

✓ Düzgün özellik; arka plan için iyi

⚠ Keskin özellikler yumuşar

GlobalAveragePooling2D

`mean(tüm H×W)`

✓ Parametre sayısını düşürür; GAP→Dense

⚠ Ince konumsal bilgi gider

GlobalMaxPooling2D

`max(tüm H×W)`

✓ En güçlü aktivasyonu alır

⚠ GAP'a göre daha az kullanılır

BatchNorm

`BatchNormalization()`

Mini-batch boyunca normalize. Eğitim/test farkı var (momentum). CNN için standart.

LayerNorm

`LayerNormalization()`

Tek örnek içinde normalize. Batch size'dan bağımsız. Transformer standart.

GroupNorm

`GroupNormalization(32)`

Kanalları gruplara bölerek normalize. Küçük batch size'larda BN'den iyi.

Data Augmentation — Neden ve Nasıl?

tf.data pipeline · Keras preprocessing layers · Albumentations

⚠ Problem: Veri Yetersizliği

Modelin eğitimde görmediği açı, aydınlatma, ölçek değişimleri test setinde başarısızlık yaratır. Daha fazla veri toplamak pahalıdır.

✓ Çözüm: Data Augmentation

Eğitim sırasında resimleri rastgele dönüştürerek yapay çeşitlilik yaratır. Modelin genelleme kapasitesi artar, overfitting azalır.

↔ RandomFlip

`RandomFlip('horizontal')`

Yatay/dikey çevirme. CIFAR-10 için yaygın. Sayılar için dikkat (6↔9).

RandomRotation

`RandomRotation(0.1)`

±36° döndürme. 0.1=tam açının %10'u. Küçük açı tercih edilir.

RandomZoom

`RandomZoom(0.15)`

±%15 yaklaşma/uzaklaşma. Nesne ölçek değişimine karşı dayanıklılık.

RandomBrightness

`RandomBrightness(0.2)`

Parlaklık rassal ayarı. Farklı aydınlatma koşullarını simüle eder.

RandomContrast

`RandomContrast(0.2)`

Kontrast değişimi. Düşük ve yüksek kontrastlı görüntülere hazırlık.

RandomCrop

`RandomCrop(28, 28)`

Rastgele kırpma. Nesnenin farklı pozisyonlarda olmasını simüle eder.

⚡ Keras Preprocessing Layers eğitim sırasında GPU'da çalışır — veri artırımı model grafiğine dahildir, .predict()'te devre dışı kalır

Augmentation Pipeline — 3 Farklı Yaklaşım

Keras Layers · tf.data map() · CutMix & MixUp (gelişmiş)

augmentation_pipeline.py

```python
# ── YÖN 1: Keras Sequential Augmenter ──────────────
augmenter = keras.Sequential([
    layers.RandomFlip('horizontal'),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.15),
    layers.RandomBrightness(0.2),
    layers.RandomContrast(0.2),
], name='augmenter')

# ── YÖN 2: tf.data map() pipeline ──────────────────
@tf.function
def augment(image, label):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_crop(image, [28,28,3])
    image = tf.image.resize(image, [32,32])
    image = tf.image.random_brightness(image, 0.2)
    image = tf.clip_by_value(image, 0.0, 1.0)
    return image, label

train_ds = (train_ds
    .shuffle(5000)
    .map(augment, num_parallel_calls=tf.data.AUTOTUNE)
    .batch(BATCH_SIZE)
    .prefetch(tf.data.AUTOTUNE))
```

⊘ Sadece Eğitimde

Augmentation YALNIZCA eğitim setine uygulanır. Validation ve test orijinal kalır — yoksa hatalı değerlendirme.

⚡ Online Augmentation

Her epoch'ta farklı dönüşüm → etkin veri artışı. model.fit() sırasında her batch farklı görünür.

⚖ Şiddet Ayarı

Çok agresif augmentation underfitting'e yol açar. Doğruluğu izle: val_acc artmıyorsa şiddeti azalt.

🔀 CutMix / MixUp

İki görüntüyü karıştırır. Etiketler de karışır (soft label). CIFAR-10'da +1-2% accuracy kazandırır.

✓ Augmentation → GPU içi işleme | num_parallel_calls=AUTOTUNE → CPU pipeline otomatik optimize | prefetch → bekleme süresi sıfır

Residual Bloklar — 'Derin Ağların Derin Sorunu'

Vanishing Gradient · Skip Connection · Bottleneck · ResNet-v1 vs v2

⚠ Derin Ağlarda Vanishing Gradient

100+ katmanlı ağlarda gradyan, geri yayılırken her katmanda küçülür. İlk katmanlara ulaştığında neredeyse sıfır olur. Ağ öğrenemez.

✓ Skip Connection (He et al., 2015)

F(x) + x → gradyan doğrudan erken katmanlara ulaşır. 'Kimlik fonksiyonu' kısa yol sağlar. 152+ katman eğitilebilir hale gelir.

Basic Block (ResNet-18/34)

```
Conv 3×3 → BN → ReLU

Conv 3×3 → BN

Add(shortcut, x) → ReLU

──────────────────

Parametre: 2 × (3×3×C×C)

Kullanım: Küçük ağlar için
```

Bottleneck Block (ResNet-50+)

```
Conv 1×1 (C→C/4) → BN → ReLU

Conv 3×3 → BN → ReLU

Conv 1×1 (C/4→C) → BN

Add(shortcut, x) → ReLU

──────────────────

Parametre ~4x az, daha derin
```

PreActResNet v2 (Önerilen)

```
BN → ReLU → Conv 3×3

BN → ReLU → Conv 3×3

Add(shortcut, x)  ← ReLU YOK

──────────────────

Gradyan akışı daha temiz

Eğitimde daha stabil
```

💡 Shortcut boyutu farklıysa: Conv 1×1 (stride=2) ile projeksiyon | ResNet-50 = 16 Bottleneck blok + stem + head

Modern CNN Mimarileri — Hangisini Seçmeli?

VGG · ResNet · EfficientNet · MobileNet · ConvNeXt

Mimari	Yıl	Parametre	ImageNet Top-1	Seçim Kriteri
VGG-16/19	2014	138M / 143M	71.5%	Büyük ama basit. Benchmark için hâlâ kullanılır.
ResNet-50	2015	25.6M	76.1%	Sektör standardı. İyi denge. Fine-tuning için ideal.
MobileNetV2	2018	3.4M	72.0%	Mobil/edge için. Çok hafif, yeterince iyi.
EfficientNetB0	2019	5.3M	77.1%	Bileşik ölçekleme. Parametre başına en iyi acc.
EfficientNetB4	2019	19.3M	82.9%	Yüksek doğruluk gerektiğinde tercih edilir.
ConvNeXt-Tiny	2022	28.6M	82.1%	Transformer'dan ilham. Modern CNN'lerin zirvesi.

Transfer Learning Rehberi:

Az veri + hızlı sonuç → MobileNetV2 | Genel kullanım → ResNet-50 | En yüksek doğruluk → EfficientNetB4+

ImageNet Top-1 değerleri yaklaşıktır; ölçme yöntemi ve eğitim detaylarına göre değişir.

Transfer Learning — 3 Strateji

Feature Extraction · Partial Fine-Tuning · Full Fine-Tuning

1 Feature Extraction (Dondurma)

▸ Pre-trained ağırlıklar dondurulur (trainable=False). Sadece yeni eklenen başlık (head) eğitilir.

📌 **Veri azsa (<1K örnek) veya kaynak domain çok benzer olduğunda. Hızlı ve stabil.**

```
base_model = EfficientNetB0(weights='imagenet',
            include_top=False, input_shape=(224,224,3))
base_model.trainable = False  # TÜM BLOKLAR DONDURULDU

x = base_model.output
x = layers.GlobalAveragePooling2D()(x)
out = layers.Dense(10, activation='softmax')(x)
```

2 Partial Fine-Tuning (Kısmi Açma)

▸ Üst N bloğu (son katmanlar) açılır, alt katmanlar donuk kalır. İki aşamalı eğitim.

📌 **Orta büyüklükte veri (1K–10K) veya domain kısmen farklıysa. Altın standart.**

```
base_model.trainable = True
# Sadece son 30 katmanı aç:
for layer in base_model.layers[:-30]:
 layer.trainable = False
# Çok küçük LR ile devam et:
model.compile(optimizer=Adam(1e-5), ...)
```

3 Full Fine-Tuning (Tam Açma)

▸ Tüm ağırlıklar güncellenir. Çok küçük LR (1e-5 veya daha az) zorunlu.

📌 **Büyük veri (>10K) veya domain çok farklıysa. Catastrophic forgetting riski var!**

```
base_model.trainable = True
# Tüm ağ açık — ÇOK KÜÇÜK LR!
optimizer = Adam(1e-5)
# Warmup + Cosine decay önerilir
# EarlyStopping şart!
model.compile(optimizer=optimizer, ...)
```

⚠ İki aşamalı eğitim: Aşama 1) head eğit (büyük LR, ~20 epoch) → Aşama 2) fine-tune (küçük LR, ~30 epoch)

Transfer Learning — Tam Uygulama Kodu

EfficientNetB0 · İki Aşamalı Eğitim · Grad-CAM Hazırlığı

transfer_learning.py

```python
# ── AŞAMA 1: Feature Extraction ──────────────────────
base = keras.applications.EfficientNetB0(
    weights='imagenet',
    include_top=False,       # Sınıflandırma başı yok
    input_shape=(96,96,3),   # CIFAR-10 büyütülmüş
    pooling=None,
)
base.trainable = False

inputs = keras.Input(shape=(96,96,3))
x = base(inputs, training=False)  # BN kapalı!
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.3)(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.2)(x)
outputs = layers.Dense(10,'softmax')(x)
model = keras.Model(inputs, outputs)
# LR: 1e-3  |  20 epoch

# ── AŞAMA 2: Fine-Tuning (son 30 katman) ─────────────
base.trainable = True
for layer in base.layers[:-30]:
    layer.trainable = False
# BatchNorm eğitimde DONDURULUR:
for layer in base.layers:
    if isinstance(layer, layers.BatchNormalization):
        layer.trainable = False
# LR: 1e-5  |  Cosine Decay
```

⚠ training=False neden kritik?

base(inputs, training=False) → BatchNorm ve Dropout katmanlar çıkarım modunda çalışır. Eğitim istatistikleri korunur.

BatchNorm'u dondurun

Fine-tuning sırasında BN katmanları dondurulmazsa istatistikler bozulur. Küçük batcih boyutuyla ciddi performans kaybı.

LR 10-100x küçük olmalı

Fine-tuning LR'si başlık eğitiminden en az 10x küçük olmalı. 1e-3 → 1e-5. Catastrophic forgetting'i önler.

Warmup + EarlyStopping

Fine-tuning başlangıcında küçük LR'den büyüğe çıkın (warmup). EarlyStopping ile val_acc düşmeye başlayınca dur.

EfficientNetB0 + 2-aşamalı eğitim → CIFAR-10'da %90+ | Sıfırdan eğitim → ~%85 | Fark: 1000 ImageNet sınıfından aktarılan desen bilgisi

Grad-CAM — Modelin 'Nereye Baktığını' Görmek

Gradient-weighted Class Activation Mapping · Yorumlanabilirlik

Grad-CAM, bir görüntüyü sınıflandırırken modelin hangi bölgelere odaklandığını ısı haritası olarak gösterir. Son konvolüsyon katmanının feature map'lerine hedefe göre gradyanlar hesaplanır, ağırlıklı ortalama alınır.

grad_cam.py

```
def compute_gradcam(model, img_array, last_conv_layer, pred_idx=None):
    # Son conv katmanına + çıkışa erişim modeli
    grad_model = keras.Model(
        model.inputs,
        [model.get_layer(last_conv_layer).output, model.output]
    )

    with tf.GradientTape() as tape:
        conv_out, preds = grad_model(img_array)
        if pred_idx is None:
            pred_idx = tf.argmax(preds[0])
        class_channel = preds[:, pred_idx]

    # Gradyanları hesapla
    grads = tape.gradient(class_channel, conv_out)
    pooled_grads = tf.reduce_mean(grads, axis=(0,1,2))
    conv_out = conv_out[0]  # Batch boyutunu kaldır
    heatmap = conv_out @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    heatmap = tf.maximum(heatmap, 0) / (tf.math.reduce_max(heatmap)+1e-8)
```

1 — Gradyanları Kaydet

tf.GradientTape ile son Conv katman çıktısını ve model çıkışını aynı anda hesapla.

2 — Global Pooling

Her özellik kanalının sınıfa katkısı = gradyanların uzamsal ortalaması.

3 — Ağırlıklı Kombinasyon

Feature map'leri kanal ağırlıklarıyla çarp, topla. Negatif değerleri ReLU ile sıfırla.

4 — Isı Haritası Çakıştır

Heatmap'i orijinal resme overlay et. Kırmızı bölgeler modelin odaklandığı yerleri gösterir.

Grad-CAM uygulamaları: hatalı tahminlerin analizi · veri kalitesi kontrolü · model güvenilirliği kanıtlama

Bugünkü Uygulamalar — 4 Python Dosyası

Sırasıyla çalıştır · Veri: CIFAR-10 · pip install tensorflow

01 CNN Temelleri & Mimari Tasarımı

`uygulama_01_cnn_temelleri.py`

- Conv2D + MaxPool + BN + GAP tam mimari
- Filtre boyutu / sayısı ablasyon çalışması
- Stride vs Pooling karşılaştırması
- Feature map görselleştirme (conv1 çıktıları)
- Parametre sayısı ve FLOPs analizi

02 Data Augmentation Stratejileri

`uygulama_02_data_augmentation.py`

- tf.data pipeline + prefetch + cache
- Keras Preprocessing Layers
- CutMix & MixUp implementasyonu
- Augmentation şiddeti ablasyonu
- Val accuracy vs augmentation karşılaştırması

03 İleri CNN: ResNet & DepthwiseSep

`uygulama_03_ileri_cnn.py`

- Basic ve Bottleneck ResBlock'u elle inşa
- PreActResNet v2 implementasyonu
- DepthwiseSeparableConv vs Conv2D karşılaştırması
- Channel Attention (SE Block) modülü
- Parametre başına accuracy analizi

04 Transfer Learning & Grad-CAM

`uygulama_04_transfer_learning.py`

- EfficientNetB0 3 strateji karşılaştırması
- 2-aşamalı eğitim (head → fine-tune)
- Catastrophic forgetting demosu
- Grad-CAM ısı haritası görselleştirme
- Doğru/yanlış tahmin analizi

```
pip install tensorflow scikit-learn numpy matplotlib opencv-python  (+ CIFAR-10 otomatik
```

Hafta 2 Cumartesi Tamamlandı!

Bugünün kazanımları:

- ✓ Conv2D, Feature Map, Pooling, GAP tam kavradık
- ✓ ResNet Basic/Bottleneck/PreAct bloklarını yazdık
- ✓ Transfer Learning 3 strateji + 2 aşamalı eğitim
- ✓ Data Augmentation pipeline + CutMix/MixUp
- ✓ DepthwiseSeparableConv — parametre tasarrufu
- ✓ Grad-CAM ile model yorumlanabilirliği

Hafta 2 Pazar: RNN · LSTM · Attention · Sequence Modeling · Zaman Serisi Tahmini