FER2013 ile CNN Tabanlı Duygu Tanıma

Transfer learning (ResNet) + GPU (CUDA) + optimizer karşılaştırması (AdamW / SGD / KarcıFANN / Momentum-Karcı)

Gri veri ⟶ 3 kanal | 224×224 giriş | Freeze ⟶ Unfreeze (layer4)

7 sınıf: angry, disgust, fear, happy, neutral, sad, surprise

Veri Seti: FER2013 (sınıf dengesizliği kritik)

Gözlem (senin çıktın):

- Görüntüler gri tonlamalı; çoğu 48×48 (küçük).
- Sınıflar dengesiz: "disgust" çok az.
- Dengesizlik, modelin "sad/neutral" gibi sınıflara kaymasına neden olur.

Sınıf sayıları (train)

Çözüm yaklaşımı

- WeightedRandomSampler (örnekleme ile dengeleme)
- Label smoothing (0.1) ve data augmentation
- Confusion matrix ile hangi sınıf kayıyor gör

Ön-işleme (gri ⟶ 3 kanal) ve augmentasyon

Neden gri → 3 kanal?

- ResNet gibi pretrained modeller RGB bekler (3 kanal).
- Gri görüntüyü 3 kez kopyalayıp (R=G=B) uyum sağlarız.
- Önemli: eğitim ve webcam tarafında aynı normalize kullanılmalı.

Augmentasyon önerisi (train)

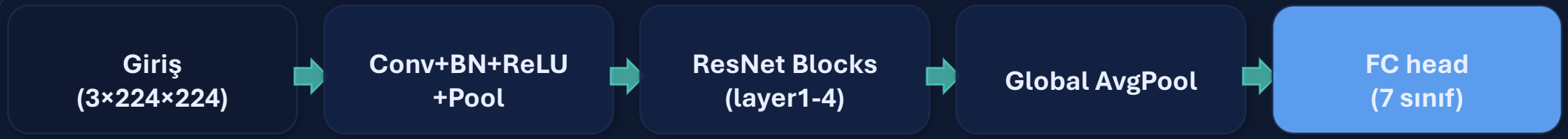
- RandomCrop / RandomResizedCrop
- HorizontalFlip
- Small rotation (±10°)
- RandomErasing (hafif)

Conv1 kod kısmı

```
def bbackboneuild_model(num_classes: int, backbone="resnet34",
dropout=0.2):
    if == "resnet18":
        m = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
        in_feats = m.fc.in_features
    elif backbone == "resnet34":
        m = models.resnet34(weights=models.ResNet34_Weights.DEFAULT)
        in_feats = m.fc.in_features
    else:
        raise ValueError("backbone must be resnet18 or resnet34")

    # 1 kanal input için conv1 değiştir
    old = m.conv1
    m.conv1 = nn.Conv2d(
        1, old.out_channels,
        kernel_size=old.kernel_size,
        stride=old.stride,
        padding=old.padding,
        bias=False
    )
```

Model: ResNet backbone + sınıflandırıcı head



Transfer learning stratejisi

- Başta backbone dondurulur (freeze): sadece head öğrenir.
- Birkaç epoch sonra layer4 çözülür (unfreeze): ince ayar başlar.
- Backbone LR düşük, head LR yüksek: daha stabil öğrenme.

Neden unfreeze?

- FER2013 düşük çözünürlüklü, domain farkı var.
- Üst katmanlar (layer4) daha task-spesifik özellik öğrenir.
- Senin loglarda unfreeze sonrası acc hızlı artıyor.

Eğitim döngüsü (AMP, grad clip, scheduler sırası)

Bir batch içinde olanlar

- Forward: logits = model(x)
- Loss: CrossEntropy (+ label smoothing / class weights opsiyonel)
- Backward: scaler.scale(loss).backward()
- Unscale + grad_clip (patlamayı engeller)
- Optimizer step (Karcı: step(loss_scalar) / diğerleri: scaler.step)
- Scheduler.step() HER ZAMAN optimizer’dan sonra

AMP (Automatic Mixed Precision) neden?

- GPU’da daha hızlı (FP16/TF32), daha az VRAM
- GradScaler: azaltır
- Karcı step’te önce unscale yapmak kritik

⚠️ Uyarı: lr_scheduler.step() önce çağrılırsa ilk LR adımı “atlanabilir”.
Doğru sıra: optimizer.step() → scheduler.step()

Optimizer karşılaştırması: AdamW / SGD / KarcıFANN

Klasikler

- SGD: basit, LR hassas; momentum + schedule ile güçlenir.
- AdamW: adaptif adım + weight decay ayrıştırma (genelde stabil).
- OneCycleLR ile LR ısınma + soğuma: hızlı yükseliş gördün.

KarcıFANN (koddaki versiyon)

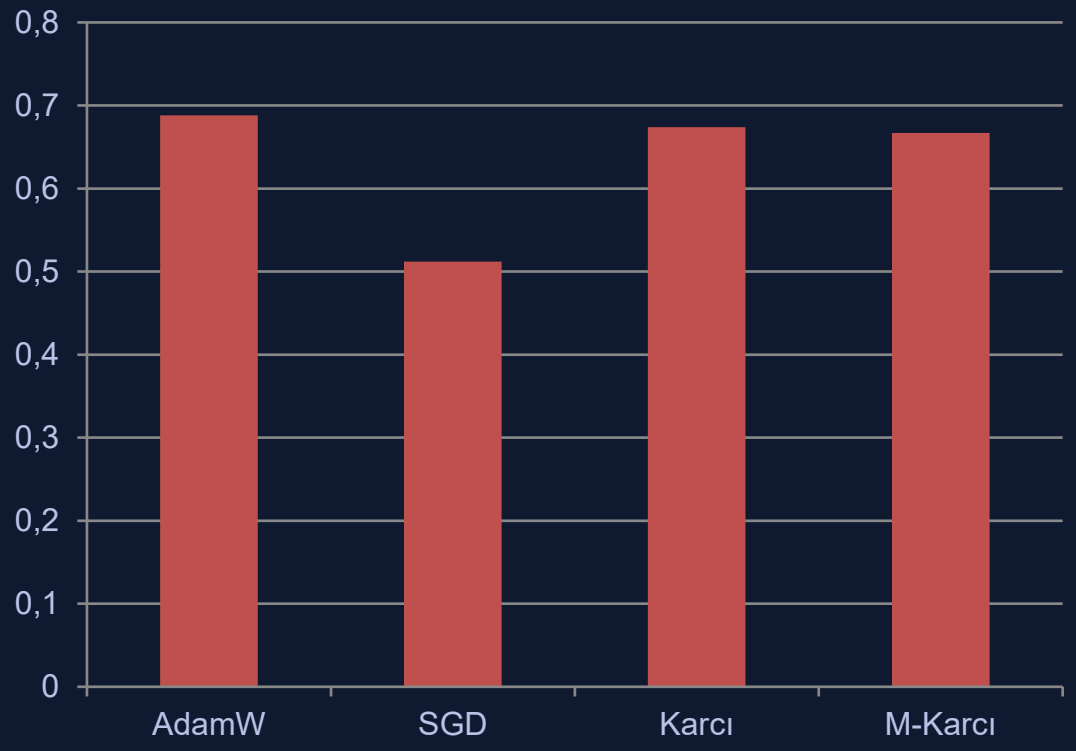
```
u = ( ((loss+ε)/(W_ref+ε))^(α−1) ) ⊙ g
W ← W − lr · clip(u)
W_ref: “pozitif referans” (negatifse önceki
pozitif değeri koru)
```

Neden bazen NaN?

- loss veya W_ref çok küçükse oran büyür → scale taşar.
- α<1 durumunda (α-1 negatif): küçük W_ref ⇒ devasa adım.
- Çözüm: eps, w_floor, ratio clip, grad clip + amp unscale.

Deney Sonuçları (örnek run)

En iyi run : AdamW ≈ 0.688 (epoch 100)



Ne işe yaradı?

- Freeze→unfreeze + farklı LR (backbone düşük, head yüksek)
- OneCycleLR ile hızlı öğrenme
- Sampler ile dengesizlik baskılama
- Label smoothing ile daha iyi genelleme

Hedef 0.85+ için (FER2013) gerçekçi mi?
Genelde FER2013'te "in-the-wild" ve düşük çözünürlük yüzünden 0.85 çok zor.
Ama 0.70–0.75 bandı güçlü ayarlarla mümkün.

Webcam testinde “hep sad” çıkması: neden & çözüm

En yaygın 3 sebep

- Normalize uyuşmuyor: train (x-0.5)/0.5 ama webcam farklı.
- Gri→3 kanal dönüşümü atlanıyor veya yanlış sırada.
- Yüz kırpma yok: arka plan baskın kalıyor (özellikle ev ışığı).

Çözüm kontrol listesi

- Webcam preprocess = train preprocess (aynı resize + normalize).
- Grayscale al → 3 kanal yapıldı → modele verildi.
- Yüz tespiti ile crop (mediapipe / opencv).

Pratik test: “tek kare” doğrulaması
1) Train loader’dan 1 örnek al, modele ver, doğru tahmin ediyor mu?
2) Aynı örneği webcam pipeline’ından geçir (dosyadan okuyup), sonuç aynı mı?

GPU performans ayarları (Windows + Jupyter)

Hız için pratik ayarlar

- device=cuda, model.to(device), batch ↑ (VRAM yettiği kadar)
- DataLoader: pin_memory=True, persistent_workers=True
- num_workers: Windows'ta 2–4 (donarsa 0)
- torch.backends.cudnn.benchmark=True
- AMP autocast + GradScaler

"RAM çok, VRAM az kullanıyor" neden?

- Dataset CPU RAM'de decode edilir (PIL/OpenCV).
- GPU VRAM: batch + model + activations kadar.
- VRAM'i artırmanın yolu: batch ↑ / img_size ↑ (ama hız düşer).

Not: Kernel/Python yolu karışırsa Jupyter CPU Torch kullanabilir.
Çözüm: CUDA Torch kurulu kernel'i seç (py310-cu118 gibi) ve "restart kernel" yap.

1) Problem

FER2013’te 7 duygu sınıfını CNN ile sınıflandırmak.

2) Yaklaşım

- Gri görüntüleri 3 kanala çevirip 224×224’e ölçekledim.
- ResNet34 backbone + yeni bir sınıflandırıcı head kullandım.
- Önce head eğitimi (freeze), sonra layer4 unfreeze ile fine-tune yaptım.

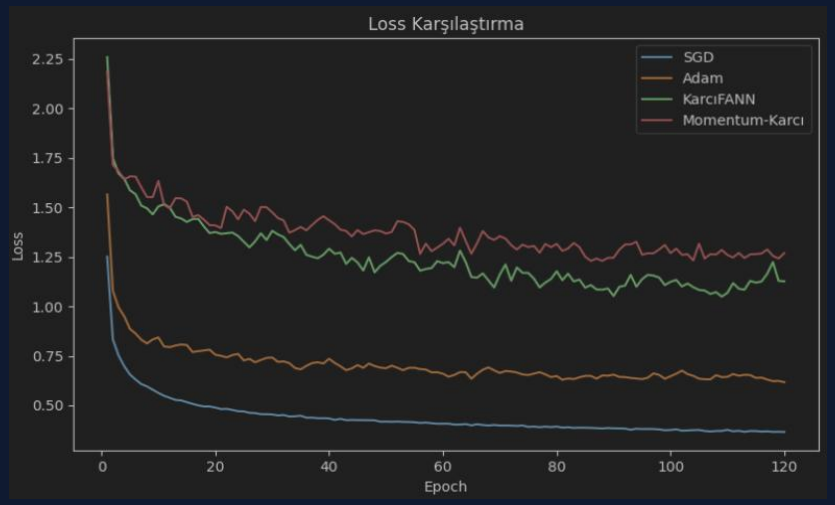
3) Deneyler & sonuç

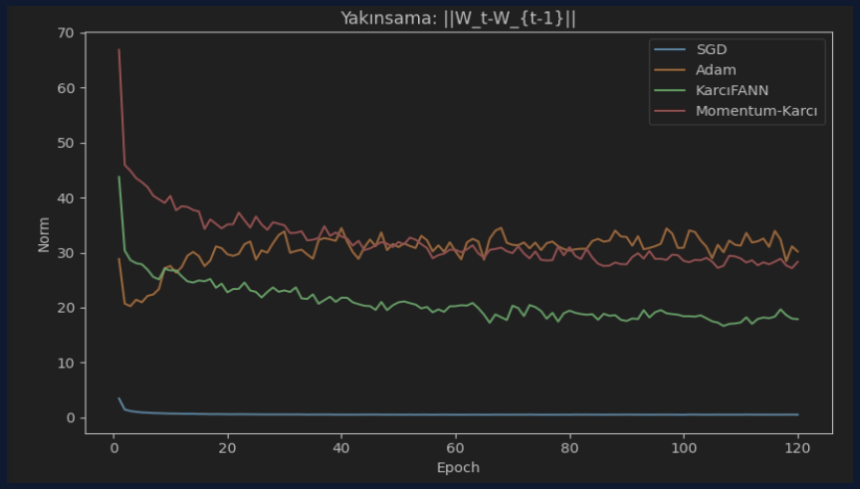
- AdamW + OneCycleLR en iyi sonucu verdi (≈0.69).
- Sampler/label smoothing ile dengesiz sınıflar daha stabil öğrenildi.
- KarcıFANN optimizasyonunda NaN riskini eps/w_floor/clip ile bastırdım.

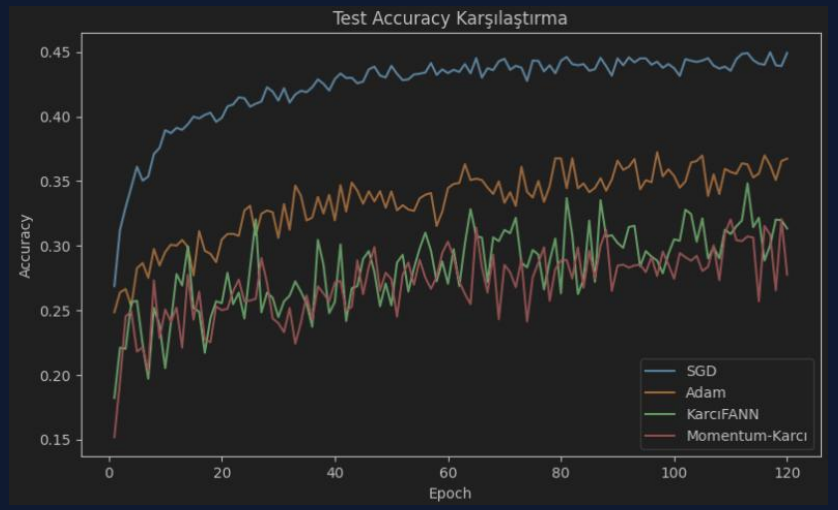
4) Sonraki adım

Daha yüksek doğruluk için: daha güçlü backbone (EfficientNet/ConvNeXt), daha iyi label (FERPlus), veya daha büyük veri (AffectNet/Aff-Wild2).

Mediapipe ile extract features yaptığımızda çıkan sonuçlar







1) SGD (Stochastic Gradient Descent)

Tek ana fikir: gradyan ne diyorsa o yönde gider.
Avantaj:
-- Basit, hızlı, az bellek.
-- Doğru LR + momentum + scheduler ile çok iyi genelleme verebilir.
Dezavantaj:
-- LR ayarı hassas. Kötü LR seçersen yavaş öğrenir veya zıplar.
-- Başlangıçta AdamW kadar hızlı toparlamayabilir.
Tek adımda sadece gradyanı kullanır, adaptif hız yapmaz.

2) AdamW

Adam'ın mantığı: her parametre için adaptif LR uygular (momentum + RMSprop karışımı).
Avantaj
-- Fine-tuning'de genelde çok güçlü.
-- LR ayarı SGD kadar hassas değil.
-- OneCycleLR ile birleşince "roket gibi" toparlar 🚀 (senin 0.68'e çıkışın buna örnek)
Dezavantaj
-- Bazen fazla hızlı ezberleyebilir (overfit), ama augmentation + label smoothing bunu dengeler.

3) KarcıFANN(Özel optimizer)

klasik optimizerlardan farklı: adımı sadece gradyana göre değil, **loss ve parametrenin referans ağırlığı (wref)** üzerinden ölçekliyor:
Kabaca:

ratio = (loss + eps) / (wref + eps)
scale = ratio^(alpha-1)
update = scale * grad (sonra clip)
w = w - lr * update

Bunun fikri ne?
Loss büyüdüyse update büyüsün, loss küçükse update küçülsün.
Parametre büyüklüğüne göre bir "normalize/adaptif adım" gibi davranıyor.
Yani Adam gibi adaptif, ama adaptiflik kaynağı farklı: **loss ve weight referansı**.
Neden bazen NaN gördü?
Bu optimizer "loss_scalar" ile çalışıyor, loss NaN olursa ratio NaN olur, her şey NaN'a gider.
Ayrıca wref güncellemen torch.where(p.data > 0, p.data, wref) gibi, negatif/0 bölgelerinde referans sıkıntılı kalabilir.
AMP + clip + ratio pow gibi işlemler NaN'a daha yatkın.
Ne zaman mantıklı?
Tez/proje özgünlüğü için güzel: "klasik optimizer yerine kendi optimizasyon şemam".
Ama pratikte stabilite ve hiperparametre ayarı daha zor.