Konvolüsyon işleminde genelde filtremizin orta noktasını giriş matrisinin her bir elemanının üstüne koymak isteriz. Bu yüzden padding işlemi zorunlu hale gelir. Ayrıca padding uygulanmazsa her defasında boyut küçüleceği için veri kaybı olabilir. Konvolüsyon işlemi giriş ve filtrenin solt üst köşesinden başlanarak da yapılabilir Genelde 3x3 ve 5x5 boyutunda filtreler seçeriz.

Pooling işlemi ile verideki önem arz eden ayırıcı elemanları seçerek eğitilecek parametre sayısını düşürürüz. Pooling işlemi ezberlemenin(overfitting) önüne geçmemize yardımcı olur. Max, Min, Mean(Ortalama) Pooling bazı pooling yöntemleridir. Tıpki konvolüsyon işleminde olduğu gibi pooling işleminde de stride(adım sayısı) vardır. Kerasta stride sayısı otomatik olarak verdiğimiz filterinin yüksekliği X genişliği kadar stride sayısı ile işlem yapar. Bunu değiştirmek için strides = (a,b) şeklinde değer verebiliriz. a = x eksenindeki adım sayısı, b = y eksenindeki adım sayısı.

Konvolüsyonel sinir ağlarında ilk konvolüsyon katmanlarında kenar, köşe, çizgi gibi bilgiler öğrenilirken sonraki katmanlarda bu bilgilerden daha genel bilgiler öğrenilir. Bunu sağlamak için max pooling işlemini kullanırız. Max pooling ile bir sonraki katmanda filtreleme(konvolüsyon) için vereceğimiz girdiden önemli olan özelliklerin seçilmesini bu sayede sonraki katmanlarda ağımızın daha genel bilgilere odaklanıp daha iyi çıkarım yapmasını sağlarız. Örneğin son konvolüsyonel katmanda bir kedinin sadece kenar bilgilerini dense layere göndermek ve doğru bir sınıflandırma yapmasını beklemek yanlış olacaktır. Oysaki Max Pooling kullansaydık daha elimizde daha genel bilgiler olacaktı. Bu sebepten ötürü Max Pooling, Pooling metodları içinde en çok tercih edilenidir.

Dropout overfitting problemini çözmeye yardımcı olur.

Çok katmanlı sinir ağlarında seçeceğimiz aktivasyon fonksiyonu türevlenebilir ve türevi sabit olmayan(nonlineer) bir fonksiyon olmalıdır. Türevlenemeyen bir fonksiyon çok katmanlı sinir ağında kullanılamaz. Türevi sabit olan(lineer) bir fonksiyon kullanırsak öğrenme çok katmanlı bir yapıda gerçekleşmez. Lineer regresyonda ise tek bir katmanımız olduğu için lineer fonksiyon kullanılır ve iş görür.

İkili sınıflandırma için loss lonksiyonu olarak binary cross entropy ilk seçenek olmalıdır. MSE'de kullanılabilir. Ama bce daha uygundur

Birden fazla sınıf varsa ve her örnek bir çıktı(sınıf) alıyorsa > Multiclassification

Birden fazla sınıf varsa ve her örnek birden fazla çıktı alıyorsa > Multi-label, Multiclass Classifacation

Sınıflandırma problemlerinde ara katmanlara çıktı katmanındaki nöron sayısından az nöron koymamak gerekli.

Multiclass classification için -> Loss fonksiyonu olarak categorical_crossentropy, son katman aktivasyon fonksiyonu olarak softmax kullanılmalıdır.

Sınıflandırma problemlerinde categorical_crossentropy loss fonksiyonu kullanıyorsak çıktı verilerimizi one-hot encoding ile kodlamamız gerekir. Yada sparse_categorical_crossentropy ile çıktıları tamsayı olarak verebiliriz. 0-100 arası tam sayılar gibi.

Multi-label multiclass classification yaparken son katmanda sigmoid fonksiyonu kullanmak gereklidir. Softmax ile olmaz. Ayrıca loss fonksiyonu olarak da binary_crossentropy kullanmak gerekli.

Verisetini normalize ederken tüm verisetini tek bir seferde normalize etmeliyiz yada eğitim veri seti ile normalize için gerekli değerleri hesaplayıp normalize işlemini yaptıktan sonra o gerekli veriler ile test setini de normalize ederiz. Yani test veri seti için tekrar normalize için gerekli değerleri(min,max,ortalama,standart sapma gibi) değerleri hesaplamayız.

Franchois Chollet'in kitabında : regresyon problemlerinde son katmanda aktivasyon fonksiyonu kullanmanın yani lineer aktivasyon fonksiyonu kullanmanın en iyi seçim olacağı söyleniyor. Ayrıca regresyon için MSE loss fonksiyonu kullanıyoruz. Metrik olarak mae'ye bakabiliriz.

Verisetini ayırırken kullanılan yaklaşımlar:

1:Eğitim-Test: Veriseti 2 parçaya bölünür. Bence kullanılması pek mantıklı değil.

2.Eğitim-Doğrulama-Test: Veriseti 3 parçaya bölünür. En çok pay eğitimdedir. Sonrasında genelde testde sonra da doğrulamadadır. Verisetimizdeki örnek sayısı fazla ise kullanılması faydalıdır. Eğitim sırasında doğrulama veriseti ile eğitimin anlık olarak kontrolü mümkündür. Ağın ezberleme yapıp yapmadığı kontrol edilir eğer ezberleme yapmaya başlıyorsa erken durdurma yapılabilir. Peki neden doğrulama ve test şeklinde 2 veri setimiz var, doğrulama yeterli değil mi ? Doğrulama veriseti ile modelimizin hiperparametrelerini(katmanlardaki nöron sayısı, katman sayısı, öğrenme katsayısı vb.) ayarlarız. Bu hiperparametreleri her değiştirdiğimizde doğrulama veri setindeki başarıma bakarız. Sürekli bu verisetini baz alarak bir eniyileme yaptığımız için her ne kadar modelimiz doğrulama veriseti ile eğitilmemiş olsa bile bir nevi bu doğrulama verisetini öğrenmiş olur. Buna bilgi kaçağı da diyebiliriz. Bu yüzden hiperparametreleri optimize edip ağımız kullanıma hazır hala geldiğinde onu test veriseti ile test ederek performasına bakarız. Ayrıca bu yaklaşımda ideal hiperparametreleri bulduktan sonra ağımızı eğitim ve doğrulama verisetlerinin birleşimi ile eğitebiliriz. Eğitim verisetinin örnek sayısı zaten daha fazla olacağı için yukarıdaki overfitting problemi oluşmaz.

3.K-fold Crossvalidation : Bu yöntemle tüm veriseti ile eğitim ve test yapılması sağlanır. Modelin eğitim sayısı k adet test metriğinin ortalamasıdır. Verisetindeki örnek sayısı az ise kullanılmalıdır. K-fold crossvalidation ile model üstünde genel bir analiz yaptıktan sonra, eğitim veri seti ile kurulan başka bir model(ağırlıkları sıfırlanmış/random atanmış) eğitilir ve test veri seti ile test edilir. Son olarak kullanılacak model budur. Sonuç olarak testte kullanılacak test verisini daha önce K-fold'da kullanmamamız gereklidir. Bu sayede yukarıda anlatılan bilgi kaçağının önüne geçeriz ve daha iyi bir son değerlendirme yapmış oluruz.

4.Karıştırılmış Döngüsel K-fold Doğrulama: K-fold crossvalidationda olduğu gibi verisetimizdeki örnek sayısı azsa kullanılır. Ona ek olarak modelimizi olabildiğince yüksek hassasiyette ayarlamamıza olanak sağlar. Diğer yöntemden farklı olarak verisetini k parçaya bölmeden önce verisetini karıştırır. Karıştırılan veri seti k parçaya bölünür. Biri validation diğerleri train için kullanılır. Bu işlem p(döngü sayısı) kez tekrarlanır. Yani sonuç olarak p*k kez bir model eğiştmiş oluruz. Bu K-fold crossvalidation için k kez idi. Dolayısıyla bu yöntem maliyeti yüksek bir yöntemdir.

Veri Setinin Karıştırılması:

Genel olarak verisetlerini karıştırarak kullanmamız faydalı olacaktır. Fakat istisna olarak geçmişteki verilerle bir modeli eğitip gelecek hakkında tahmin yapmak istiyorsak verimizi karıştırmamalıyız. Bu durum train/validation/test veri setleri için de geçerli. Train verisetindeki veriler validation ve test veri setinden önceki veriler, validation veri setindeki veriler de test verisetindeki verilerden önceki veriler olmalıdır.

Çıktıların Ölçeklenmesi:

Sınıflandırma problemleri için çıktıların ölçeklendirilmesine zaten gerek yoktur. Encoding(one-hot,label gibi) yeterlidir. Regresyon problemlerinde ise tek çıktılı bir ağı ele alırsak son katmanda kullanılan aktivasyon fonksiyonu eğer sigmoid(çıktıyı 0-1 arasına sıkıştırır) ise min-max minimizasyonu

yaparak çıktıyı 0-1 arasına sıkıştırırız. Çünkü hata hesaplarken y_test verilerimiz ölçeklenemiş y_test_pred değerlerimiz ise ölçeklenmiş olacak ve burada bir sıkıntı ortaya çıkacaktır. Fakat genel olarak regresyon problemlerinde son katmanda aktivasyon fonksiyonu kullanmamak daha iyi olacaktır. Bunun yanında birden fazla çıktısı olan regresyon yapan bir modelimiz varsa çıktıların ölçeklenmesi gerekebilir. Örnek olarak çıktılardan biri arabanın yaşını(0-20 arası olabilir) belirtirken diğeri de arabanın yapmış olduğu kilometreyi(0-50000 arası olabilir) gösteriyor olsun. Loss fonksiyonu bu çıktıların hatalarını hesaplayıp aktivasyon fonksiyonu eniyilemeye çalışırken her güncelleme için hesaplanan ağın toplam çıktısı fazla çıkabilir. ToplamHata = TümÇıktılarınHatalarıToplamı gibi hesaplamalar vardı. Bu durumda çıktıları da ölçeklemeliyiz.

L1 ve L2 Regularizasyonu:

Ağımızın katmanlarındaki entropiyi(düzensizliği) azaltmaya yarar. Overfitting probleminin çözümüne yardımcı olur. Bu yöntemin uygulandığı modeldeki eğitim sırasında ortaya çıkan loss değeri genel olarak(tüm epochları hesaba katarak/epoch sayısı birkaç taneden fazla olduğunda) bu yöntemin uygulanmadığı modele göre daha fazladır. Fakat validation ve test kısmında bu yöntemi uygulamak başarımı arttırabilir. Bu yöntemde bir işi daha az karmaşık ve daha basit bir yöntemle yapmak en iyi yoldur mantığı uygulanmaktadır. Bu mantık ağın ağırlıkları sıfıra yakın bir şekilde tutularak yapılır.

L1 ve L2 normlardır. L1 de lineer, L2de karesel bir durum söz konusudur. Genel olarak L2 Regularizasyonu kullanılması tercih edilir:

$$Loss = Error(y, \hat{y})$$

Normal Şartlarda Loss Fonksiyonu

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

L1 Normalizasyonu yapılmış

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

L2 Normalizasyonu yapılmış

Buradaki lambda(**λ**) işaretimiz sabit bir katsayı. Bunu biz belirliyoruz. 0.001 gibi küçük değerler alıyor.

Dropout(İletim Sönümü):

Bir regularizasyon tekniğidir. Aşırı uydurmayı engeller. Bu yöntem bir katmana uygulandığı eğitim sırasında yöntemin uygulandığı katmanın çıktılarının bir kısmı 0 değerini alır yani bir nevi bu nöronları iptal etmiş oluruz. İptal edilen bu nöronlar tasarımcı tarafından belirlenen bir orandadır. Genelde 0.2-0.5 arası seçilir. Hangilerinin 0 yapılacağı ise bir eşik değeri belirlenerek, rastgele olarak veya başka yöntemlerle belirlenir(Bundan emin değilim).

Eğitim sırasında bazı nöronları iptal ettikten sonra ya test zamanında çıktıları belirlenen dropout oranında küçültürüz yada yine eğitim sırasından iptal etmeden sonra kalan çıktıları dropout oranında büyütürüz. Genelde 2.yöntem tercih edilir. Kerasda dropout uygulamak istediğimizde, dropout uygulamak istediğimiz katmandan sonra bir dropout katmanı ekleriz. Burada sadece dropout oranını vermemiz yeterlidir, diğer işlemleri keras halleder.

Transfer Learning

1-Öneğitimli Evrişimli Sinir Ağı Kullanmak(Öznitelik Çıkarımı)

Konvolüsyonel sinir ağı modellerinde başarımı arttırmak için daha önceden eğitilmiş bir modeli kullanarak kendi problemimizi çözebiliriz. Bu yaklaşım özellikle problemimiz için elimizde bulunan veri seti azsa başarımı arttırabilmektedir. Ayrıca bu işlemi verisetimiz az olduğu için Data Augmentaion yapmak başarımı arttırmamızı ve overffitingi bir nebze engellememizi sağlayacaktır. Elimizdeki problemi çözmek için bulunan veri boyutu ile kullanacağımız öneğitimli modelin eğitildiği veri boyutuna bağlı olarak öneğitimli modelin kullanacağımız kısımlarını seçeriz. Konvolüsyonel sinir ağı modellerimiz konvolüsyonel kısım ve tam bağlı kısım(fully connected layerlar veya diğer adıyla dense layerlar)dan oluşmaktadır. Konvolüsyonel kısımdaki konvolüsyonel katmanların ilkinde renk,kenar,köşe gibi daha basit ve genel bilgiler çıkarılırken sonraki katmanlarda daha lokal bilgiler(örneğin bir kedinin burnu,gözü gibi) çıkarılır. Tam bağlı kısımda ise çok daha lokal bilgiler vardır. Bu bilgiler çıktımızın olasıkları(classification yaptığımız için) hakkında bilgi verir. Dolayısla bu kısımdaki bilgiler probleme özeldir.

Bizim çözmek istediğimiz problemdeki sınıflar kullanacağımız öneğitimli modelde çıktılarda varsa sadece elimizdeki probleme özgü veriler ile bu modelin sadece tam bağlı katmanını tekrardan(öneğitimli modelin ağırlıkları ile) eğitebiliriz. Bunun nedeni daha önceden bahsettiğim gibi konvolüsyon katmanında, tam bağlı katmandan farklı olarak daha genel(ilk katmanlarda her nesne olabilecek kenar/renk geçişi gibi bilgiler, son katmanlarda biraz daha lokal bilgiler) bulunmasıdır. Ayrıca nesnenin konumunun önemli olmadığı durumlarda, öneğitimli ağımızın çıktıları içinde bizim problemimize ait sınıflar olduğunda(eğitimde kullanılan verilerin de benzer olduğunu varsayarak) tam bağlı katmanı da alarak öneğitimli ağdan bütün olarak faydalanmamız mümkündür.

Elimizdeki problem ile öneğitimli modelin sınıfları(çıktıları) birbirinden farklı ise bu durumda oluşturacağımız modele öneğitimli modelin tam bağlı kısmını eklemeyiz. Sadece konvolüsyon kısmını alırız. Bu durum sınıflar birbirinden farklı olsa da benzer olduğu durumlarda(örneğin araba-kamyon gibi) tercih edilmelidir. Her iki modeldeki verilerin birbiriyle hiçbir benzerliği yoksa/çok azsa bu durumda konvolüsyon kısmının da ilk katmanlarını almamız gerecektir. Bu katmanlarda problemden bağımsız genel bilgilerin çıkarıldığını söylemiştim. Bu paragrafta anlattığım 2 yöntemde de öneğitimli ağdan miras aldığımız katmanları yeni modelimizde eğitmeyiz. Bunun sağlamak için bu katmanları dondururuz. Kerasta katmanObjesi.Trainable = False işlemi ile bunu sağlayabiliriz.

2-Hassas Ayar(Fine Tunning)

Öznitelik çıkarımının tamamlayıcısıdır. Hassas Ayar yapmak için için öncelikle öznitelik çıkarımının yapılmış olması gereklidir. Öznitelik çıkarımında yeni oluşturduğumuz modeldeki yeni eklediğimiz kısımlarla birlikte, öneğitimli ağdan aldığımız kısımların bazı katmanlarının çözülerek birlikte eğitilmesi ile gerçekleştirilir.

Adımları:

1. Öneğitimli bir ağdan istediğiniz parçaları alın ve ona kendi katmanlarınızı(fully connected layers,convolution layers) gibi ekleyerek bir yeni model oluşturun.
2. Öneğitimli modelden(baz model) alınan kısmı dondurun.
3. Yeni modelinize eklediğiniz kısmı eğitin.
4. Baz modelden bazı katmanları çözün.(eğitilebilir hale getirin.)
5. Yeni eklenen ve baz modelden çözülen katmanları beraber eğitin.

Buradaki ilk 3 adım öznitelik çıkarımıdır. 4.adımda bahsedilen çözülecek katmanları konvolüsyonel kısım için son katmanlar olarak seçeriz. Çünkü konvolüsyonel kısmın ilk katmanında genel bilgiler öğrenilir, bu bilgiler de zaten işimize yarayacağı için bunlara dokunmayız. Ama son katmanlara doğru probleme özgü bilgiler öğrenildiği için bunları çözüp tekrar eğitmek yani fine tunning yapmak isteriz. Bunun yanında eğer kendi problemimizdeki veri sayısı, öneğitimli modelin eğitildiği veri sayısından çok daha az ise son katmanlara finetunnig yapmak aşırı öğrenmeye(overfitting) sebep olacaktır. Sonuncu adımı gerçekleştirirken öğrenme oranımızı 10 üzeri -5 gibi çok küçük sayılar seçmeliyiz ki çözülen yani finetunnig yapılan katmanlardaki değişimler çok büyük olmasın. Eğer böyle yapmaz isek bu bilgilere zarar gelebilir.