KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

PROJENİN KONUSU

C++ İLE CNN KODLANMASI

BİTİRME PROJESİ

Adı SOYADI

330104 ONUR ERDAŞ

340824 EMRE ÖZDEMİR

2019-2020 BAHAR DÖNEMİ

ÖNSÖZ

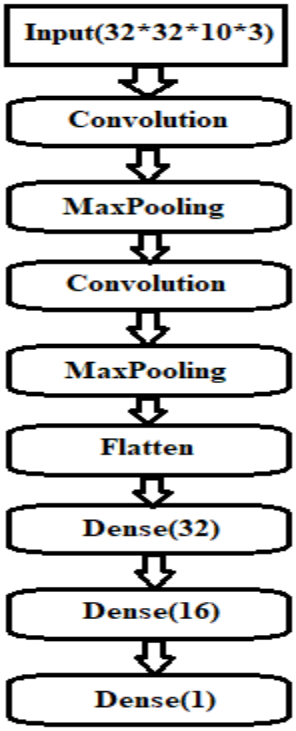
Bu uygulamada Python programlama dilinde Yapay Sinir Ağları kullanılarak bir eğitim yapılmıştır. Bu eğitim sonucunda oluşturulan ağırlıklar.*hdf5* uzantılı bir dosya olarak kayıt edilmiştir. Oluşturulan bu hdf5 uzantılı ağırlık dosyası okunup eğitim sonucunda elde edilen ağırlık değerleri bir txt dosyasına aktarılmıştır. Bu aşamadan sonra her ağ katmanının ağırlık değerleri ayrı ayrı txt dosyalarına kayıt edilmiştir. Elde edilen bu ağırlıklar C++ kısmında oluşturulan bir class yapısı aracılığı ile okunmuştur. Python tarafında kurulan CNN ağının modellenmesi amacıyla *Conv3D* adında bir class oluşturulmuştur ve CNN katmanlarının modellenmesi amacıyla düzenlenmiştir. CNN katmanlarından sonraki kısımda ise Full Connected kısmı gelmektedir. Bunun için ise nöronları modellemek amaçlı *Neuron* isminde bir class yapısı oluşturulmuştur. Bütün bu hazırlıklar tamamlandıktan sonra ise ağın kodlanması gerçekleştirilmiştir.

İçindekiler

1. Python ile CNN Ağı Modeli

1.1. Python ile Kurulan Ağ modeli

Python ile CNN ağının kurulması için bir Yapay Sinir Ağları framework'ü olan *Keras* kullanılmıştır. Bu kütüphane yardımıyla hem CNN katmanları hem de Yapay Sinir Ağları kolay şekilde kodlanmaktadır. Oluşturulan CNN ağ modeli aşağıdaki gibidir.



Şekil-1 CNN Katmanları

Şekilde görüldüğü üzere input olarak 32*32*10*3 boyutunda bir video oluşturmaktadır. Bu video sırasıyla *Conv>Pooling>Conv>Pooling* katmanlarından geçtikten sonra *flatten* ile ağa verilmektedir. Flatten'ın görevi Convolution ve MaxPooling katmanlarından geçtikten sonra elde edilen çok boyutlu matrisin tek boyuta indirgenmesini sağlamaktır. Yapay Sinir Ağı'nın ilk katmanında 32 adet nöron bulunmaktadır. Daha sonra bu giriş nöronlarının ardından 16 nöronluk bir ara katman gelmektedir. Son katmanda ise çıkış nöronu olmaktadır. Çıkış katmanı ise 1 nöron ile temsil edilmektedir. Bu çıkış katmanının aktivasyon fonksiyonu *sigmoid* olarak ayarlanmıştır. Çünkü bu ağda iki sınıflı bir tanıma yapılmaktadır. İkiden fazla çıkış bulundurmak istiyorsak

eğer bu çıkış nöronlarını 1 değil de ne kadar istiyorsak onu yazmalıyız. Ayrıca aktivasyon fonksiyonunu da değiştirmek gerekmektedir. Çok sınıflı yapılarda *softmax* fonksiyonu kullanılmalıdır. Çünkü bu fonksiyon bu amaçla oluşturulan bir fonksiyondur.

Bu katmanların Python kısmında kodlanması aşağıdaki gibi gerçekleştirilmektedir.

```
input_shape = (32, 32, 10, 3)
normalizasyon_katsayisi = 0.005

model = keras.models.Sequential()
model.add(Conv3D(8, kernel_size=(5, 5, 3), activation="relu",
kernel_regularizer=keras.regularizers.l2(normalizasyon_katsayisi),
input_shape=input_shape))

model.add(MaxPool3D((2, 2, 1), strides=(2, 2, 1)))

model.add(Conv3D(16, kernel_size=(5, 5, 3), activation="relu",

kernel_regularizer=keras.regularizers.l2(normalizasyon_katsayisi)))

model.add(MaxPool3D((2, 2, 2), strides=(2, 2, 2)))

model.add(Flatten())

model.add(Dense(32, activation='relu',
kernel_regularizer=keras.regularizers.l2(normalizasyon_katsayisi)))
model.add(Dropout(0.6))

model.add(Dense(16, activation='relu',
kernel_regularizer=keras.regularizers.l2(normalizasyon_katsayisi)))
model.add(Dropout(0.6))

model.add(Dense(1, activation='sigmoid'))
```

Bu ağ yapısını açıklayacak olursak ilk katmanda bir Convolution ile karşılaşılmaktadır. Bu Convolution katmanı 8 derinliğe sahiptir. Bunu Conv3D(8, kernel_size=(5, 5, 3)) kısmında görülmektedir. Ayrıca bu katman ve diğer katmanlarda dahil aktivasyon fonksiyonu olarak *relu* kullanılmıştır. Convolution katmanında dikkat edilmesi gereken diğer parametre ise kernel size parametresidir. Bu kernel size 3 boyutlu bir küp şeklinde matris olarak hayal edilebilir. Bu küp ağa verilen input üzerinde gezecek olan filtre matrisinin boyutlarıdır. Sırasıyla *width, height, depth* değerlerini ifade etmektedir. Yani 5*5*3 boyutunda bir matris giriş matrisi üzerinde gezdirilmektedir. Ayrıca bu katmanda genellikle kullanılan bir parametre daha vardır. Bu parametre ise *strides* parametresidir. Bu ise kernel matrisinin input matrisi üzerinde gezerken

adım sayısının ne olacağına karar verir. Convolution katmanında genel olarak (1,1,1) olarak kullanılır. Sırada ise MaxPooling katmanı vardır. Bu katmana verilen giriş strides parametresine verilen değere göre küçülmektedir. Örneğin (2,2,1) olarak verilirse giriş matrisinin width ve height değerleri yarı yarıya düşmektedir. MaxPooling'te de bir kernel matrisi vardır. Bu kernel matrisi aynı şekilde kendisine verilen input üzerinde gezmektedir.

CNN katmanlarından sonra flatten katmanı gelmektedir. Bu katmanda CNN ağından çıkan çok boyutlu matris değerinin tek boyuta indirgenmesi yapılmaktadır. Bunun amacı ise CNN katmanından çıkan matrisin direkt olarak ağa verilmesini sağlamaktır. Örneğin 4*4*4 boyutunda bir matris çıktıysa CNN katmanlarından, flatten katmanında bu matris 64 boyutunda tek boyutlu bir matrise dönüştürülmektedir ve bu şekilde ağa verilmektedir. Yapay sinir ağları kısmında ise ağ eğitilmektedir. Bu eğitim sonucunda elde edilen ağırlıklar *hdf5* uzantılı bir dosyaya kayıt edilmektedir.

Test işlemini yaparken dikkat edilmesi gereken bir durum var. Ağda bulunan aktivasyon fonksiyonlarını kaldırıp o şekilde test yapılmaktadır. Ağ katmanlarında bulunan *activation* parametresine "relu" aktivasyon fonksiyonu görülmektedir. Ayrıca ağın son nöronunda aktivasyon fonksiyonu olarak "sigmoid" kullanılmıştır. Test işleminde bunların kaldırılmasına dikkat edilmesi gerekiyor.

Convolution katmanlarında *padding* adında bir parametre vardır. Bu parametreyi "same" olarak atanması durumunda Convolution katmanına verilen bir girdinin boyutu hiç değişmeden çıkmaktadır. Bu uygulama Convolution katmanına giren bir girdinin kernel matrisinin boyutuna göre değiştiği göz önüne alınarak yazılmıştır. Bu yüzden Convolution katmanında padding parametresi değiştirilmemelidir.

1.2. Ağırlıkların txt Dosyasına Kayıt Edilmesi

Ağırlıklar eğitim tamamlandıktan sonra hdf5 uzantılı olarak kayıt edilmişti. Bu uzantılı dosyanın direkt olarak okunması imkansızdır. Bu yüzden Python ile bu dosyanın okunarak txt'ye çevirilmesi gerekmektedir. Bu amaçla bir Python kodu yazılmıştır. Bu kod eklerde sunulacaktır.

```
def getWeightsForLayer(layerName, fileName)
```

Üstteki fonksiyonla *layerName* parametresine okunması istenen katmanın ismi verilmektedir. Örneğin CNN katmanının okunması isteniyorsa buraya *conv3d* yazılmalıdır. Eğer 2 boyutlu bir CNN ağında çalışılıyorsa buraya *conv2d* yazılmalıdır. Yapay sinir ağlarının elde edilmesi isteniyorsa layerName parametresine *dense* yazılmalıdır. *fileName* parametresine ise ağırlıklarının kaydedildiği hdf5 uzantılı ağırlık dosyasının ismi verilmelidir. Böylelikle istenilen katmanlardaki ağırlıklar txt dosyasına çevrilmektedir.

2. CNN Yapısının C++ ile Kodlanması

2.1. Ağırlıkların Elde Edilmesi

Python'da elde edilen ağırlıklar C++'da kullanılması için parçalarına ayrılmıştır. Her bir katmanın ağırlıklarının kaç adet olduğu hesaplanarak bütün ağırlıkların bulunduğu txt dosyasından sırasıyla alınmıştır. İlk katman ele alınacak olursa, ilk katmanın kernel size'ı 5*5*3 olarak belirlenmiştir.

Ağa gelecek olan verinin derinliği yani input derinliği 3 boyutlu olduğundan sonraki parametre de 3 olarak belirlenmiştir. Son olarak da Convolution katmanının derinliği 8 olduğu için dizinin son boyutu 8 olarak belirlenmiştir. Kaydedilen ağırlıkların ilk 8 tanesi ilk katmanın bias değerini vermektedir. Bu 8 değeri Convolution katmanının derinliği demektir. İlk 8 ağırlıktan sonra gelen kısım ise üstte elde ediliş yöntemi belirlenen şekilde hesaplanarak alınmaktadır. Yani ilk 8 tanesini bias alarak aldıktan sonra 5*5*3*3*8 = 1800 tanesini de Convolution katmanının ağırlıkları olarak alınmaktadır.

İkinci Convolution katmanı için ise aynı hesaplamalar geçerlidir. Fakat bir önceki Convolution katmanının derinliği göz önüne alınmalıdır. Çünkü ilk katmandan çıkan dizinin dizi ikinci katmanın girdisi olmaktadır. Yani bu örnek göz önüne alınırsa ilk katman derinliği 8, ikinci Convolution katmanının derinliği ise 16'dır. Kernel size ise aynı şekilde 5*5*3'tür. Yine aynı şekilde ilk 16 adet değer ikinci Convolution katmanının bias değeri olmaktadır. Sonraki 5*5*3*8*16 = 9600 adeti ise ikinci Convolution katmanının ağırlık değerleri olmaktadır. Bu hesaplamalar ile bütün ağırlıkların bulunduğu txt dosyası tek tek katmanların ismiyle parçalarına ayrılmıştır. Daha sonra oluşturulacak olan *Conv3D* sınıfı ile bu ağırlıklar okunacaktır.

CNN katmanlarının ağırlıkları okunduktan sonra sırada FC katmanından sonra gelen Yapay Sinir Ağı katmanları gelmektedir. Bu ağırlıklar ise farklı bir txt dosyasına aktarılmıştır. Bu kısımda da hesap yapılacak olursa ağ katmanlarında bulunan nöron sayıları önem arz etmektedir. İlk katman için hesap yapılacak olursa, CNN katmanları uygulandıktan sonra elde kalan dizi 5*5*3*16 = 1200 boyutundadır. Bu dizi flatten katmanından sonra tek boyutlu 1200 elemanlı bir dizi olmaktadır. Sinir ağının ilk katmanındaki nöron sayısı 32'dir. Dolayısıyla

ağırlıkların bulunduğu txt dosyasındaki ilk 32 adet ağırlık ilk katmanın bias değerini oluşturmaktadır. Ağın ağırlık değerleri, girdi boyutu * nöron sayısı yani 1200*38 = 38400 adet ağırlık olarak hesaplanıp, bias değerlerinden sonraki 38400 adet ağırlık ilk katmanın ağırlık değerlerini oluşturmaktadır. İkinci katman göz önüne alınırsa, ilk katmanın 32 nöronundan çıkan değerler ikinci katmanın giriş değeri olmaktadır. İkinci katmanını nöron sayısı 16'dır. Dolayısıyla kalan ağırlık değerlerinin ilk 16 adeti ikinci katmanın bias değerini oluşturmaktadır. Ağın ağırlık değerleri girdi boyutu * nöron sayısı yani 32*16 = 512 olarak hesaplanıp bias değerlerinden sonraki 512 adet ağırlık ikinci katmanın ağırlık değerlerini oluşturmaktadır. Yapay Sinir Ağının ağırlıkları da bu şekilde bulunmaktadır.

2.2. Ağırlıkların Sınıf Yapısı İle Okunması

Ağırlıklar katman katman parçalarına ayrıldıktan sonra bu ağırlıkların C++'ta kullanılması için okunması gerekiyordu. Convolution ağırlıklarını okumak için *Conv3D* adında bir sınıf oluşturulmuştur. Kullanım ise aşağıdaki şekildedir.

```
Conv3D conv3d;
conv3d.setLayerArrayAllocate(5, 5, 3, 3, 8);
conv3d.setLayerWeights("conv3d_kernel_1.txt", 5, 5, 3, 3, 8);
conv3d.setBias("conv3d_bias_1.txt", 8);
```

İlk olarak Conv3D sınıfından conv3d adında bir nesne oluşturulmuştur. Bu nesnenin içinde bulunan ağırlıkların tutulmasını sağlayan 5 boyutlu bir dizi için bellekte yer açılmıştır. Bunun için *setLayerArrayAllocate()* fonksiyonu kullanılmıştır. İçindeki parametrelerin nasıl elde edildiği bir üstteki başlıkta anlatılmıştır. Daha sonra ağırlıkların setlenmesi için *setLayerWeights()* fonksiyonu kullanılmıştır. Bu fonksiyon ilk parametresi o katmanın ağırlıklarının bulunduğu txt dosyasını almaktadır. Sonraki parametreler ise bir üst satırda yer ayrılan dizinin boyutunu temsil etmektedir. Son fonksiyonda ise bias değerlerinin okunup değerlerinin ilgili diziye atanması gerçekleştirilmiştir.

İkinci katmanın ağırlıkları da aşağıdaki gibi elde edilmiştir.

```
Conv3D conv2_3d;
conv2_3d.setLayerArrayAllocate(5, 5, 3, 8, 16);
conv2_3d.setLayerWeights("conv3d_kernel_2.txt", 5, 5, 3, 8, 16);
conv2_3d.setBias("conv3d_bias_2.txt", 16);
```

CNN katmanında bulunan ağırlıklar elde edildikten sonra Yapay Sinir Ağının ağırlıkları elde edilmiştir. Bunun için de *Weights* adında bir sınıf yapısı oluşturulmuştur. Bu sınıf yapısıyla Yapay Sinir Ağında bulunan ağırlıklar C++ tarafında okunmuştur. Kullanım şekli ise aşağıdaki gibidir:

```
Weights w1(32, 38400);
w1.setLayerBias("conv3d_fc_bias_1.txt");
w1.setLayerWeight("conv3d_fc_kernel_1.txt");
```

Öncelikle Weights sınıfından *w1* adında bir nesne oluşturulmuştur. Bu w1 nesnesinin kurucu fonksiyonunda kullanılacak olan dizilerin boyutları belirlenmiştir. Kurucu fonksiyona verilen parametrelerin nasıl elde edildiği üstteki başlıkta anlatılmıştır. Bu kurucu fonksiyonun ilk parametresi bias dizisinin boyutunu diğer değeri ise ağırlıkların tutulduğu dizinin boyutunu vermektedir. W1 nesnesi oluşturulduktan sonra *setLayerBias()* ve *setLayerWeight()* fonksiyonlarına hangi txt dosyalarını okuyacakları söylenerek bias ve ağırlık değerlerinin alınması işlemi gerçekleştirilmiştir.

Diğer katmanların ağırlıklarının okunması aşağıdaki şekilde gerçekleştirilmiştir.

```
Weights w2(16, 512);
w2.setLayerBias("conv3d_fc_bias_2.txt");
w2.setLayerWeight("conv3d_fc_kernel_2.txt");

Weights w3(1, 17);
w3.setLayerBias("conv3d_fc_bias_3.txt");
w3.setLayerWeight("conv3d_fc_kernel_3.txt");
```

2.3 Nöronlar İçin Bir Sınıf Yapısının Oluşturulması

Nöronların değerlerini saklamak amaçlı bir sınıf yapısı oluşturulmuştur. Bu sınıfa Neuron adı verilmiştir. Bu sınıfta Bağlı Listeler kullanılmıştır. Nöronların değerlerinin hesaplanmasının ardından her nörona bir id değeri vererek sıralı liste şeklinde saklanmıştır. İstenen nöron değerini elde etmek amaçlı bu id özelliği kullanılmıştır.

Neuron sınıfında aşağıdaki özellikler bulunmaktadır.

```
class Neuron
{
private:

    int id;
    float value;
    Neuron* next;
    Neuron* head;
…
}
```

Bu sınıfta 2 adet fonksiyon bulunmaktadır. İlk fonksiyon *setValue()* fonksiyonudur. Bu fonksiyon yardımıyla hesaplanan nöron değeri ilgili nörona aktarılır. İkinci fonksiyon ise *getValue()* değeridir. Bu fonksiyon içine aldığı id değerine göre istenen nöronun değerini getirmektedir.

2.4. CNN KODLANMASI

Bu ağı kodlamadan önce bu ağa verilecek olan videoyu sayısal hale dönüştürmek gerekmektedir. Bu yüzden bu işlemi Python'da yapıp değerleri bir txt dosyasına her değer bir satıra gelecek şekilde kayıt edilmelidir. Bu işlemden sonra C++'da bu txt dosyası okunup bir matrise aktarılmalıdır. Bu amaçla öncelikle bir dizi oluşturulup ve bu diziye bellekte yer açtıktan sonra bir fonksiyon yazarak bu diziye videonun sayısal değerlerinin aktarılması sağlanmıştır.

```
float**** input = arrayAllocate3d(32, 32, 10, 3);
readInput(input, "c3d_input.txt");
```

Üstte görüldüğü üzere 32*32*10*3 boyutunda bir diziye bellekte yer açılmıştır. Daha sonra ağa verilecek videonun sayısal değerlerinin bulunduğu *c3d_input.txt* dosyası ve oluşturulan input dizisi *readInput* fonksiyonuna parametre olarak verilmiştir. Bu fonksiyona ikinci parametre olarak verilen txt dosyasındaki verileri, ilk parametre olarak verilen diziye aktarmaktadır.

Giriş değeri okunduktan sonraki adım ilk katman olan Convolution katmanındaki ağırlıkların okunmasıdır. Aşağıda bu ağırlıklar okunmaktadır.

```
//FIRST CONVOLUTION LAYER
Conv3D conv3d;
conv3d.setLayerArrayAllocate(5, 5, 3, 3, 8);
conv3d.setLayerWeights("conv3d_kernel_1.txt", 5, 5, 3, 3, 8);
conv3d.setBias("conv3d_bias_1.txt", 8);
```

Bu ağırlıklar okunduktan sonra, giriş dizisi olan input dizisine Convolution uygulanacaktır. Bu Convolution uygulandıktan sonra giriş dizisinin boyutları değişmektedir. 32*32*10*3 olan giriş dizisinin boyutu 28*28*8*8 boyutuna indirgenmektedir. Burada 28*28*8 video boyutunu temsil etmektedir. Sondaki 8 ise o ağdaki nöron derinliğini ifade etmektedir. Convolution katmanındaki kernel size(5,5,3) olduğundan, videonun width ve height değerleri kernelx -1, kernely -1 ve kernelz – 1 kadar küçülmektedir. Burada da width,height ve depth değerleri kernel boyutunun x, y ve z değerlerinin bir eksiği kadar küçülmüştür. Convolution fonksiyonunun prototipi aşağıdaki gibidir:

```
float**** convolution(float**** input, Conv3D conv3d, int noron_size, int frame_size, int
width, int  height, int depth )
```

Bu fonksiyona aşağıdaki şekilde kurulan ağ yapısına uygun parametreler gönderilerek input değerine Convolution uygulanması sağlanmıştır.

```
float**** result_conv = convolution(input,conv3d,8,10,32,32,3);
```

Convolution işleminin uzun bir kodlanması olduğundan bir fonksiyon haline getirilmiştir. Bu kodun okunurluğunu daha da artırmıştır. Bu fonksiyon ilk aldığı parametre kendisine verilecek olan input değeridir. Sonraki parametre ise Convolution katmanında elde edilen ağırlıklardır. Python'da kurulan ağa göre elde edilen noron_size, frame_size, width, height ve depth değerleri bu fonksiyona parametre olarak verilmektedir. Verilen bu parametrelere göre fonksiyon input dizisine Convolution uygulamaktadır. Bu değerler içeride bulunan for döngülerinin sınırlarını oluşturmaktadır. Kodun uzun haline verilen ekten bakılabilir.

Convolution uygulandıktan sonra ağ yapısında kurulan sıraya göre bakıldığında MaxPooling işlemi gelmektedir. MaxPooling için de bir fonksiyon yazılmıştır. Bu fonksiyonun prototipi aşağıdaki gibidir:

```
float**** maxPooling(float**** result_conv,int noron_size, int frame_size, int width,
int height, int mask_depth )
```

Bu fonksiyona bir input değeri verilmektedir. Bu verilen input değerine bir önceki katman olan Convolution katmanından çıkan input'tur. Bu input dizisinin boyutları maxPooling fonksiyonuna parametre olarak verilmektedir. Ayrıca Python'da kullanılan ağda MaxPooling katmanında bulunan parametreler de verilmektedir. Python'da kurulan ağda MaxPooling katmanında bulunan kernel matrisinin derinlik parametresi her katmanda değişmektedir. Bu yüzden bu derinlik değeri fonksiyondan parametre olarak alınmaktadır.

MaxPooling fonksiyonu, verilen input matrisinin boyutunu yarı yarıya küçültmektedir. Input değerinin ilk boyutu 28*28*8*8'dir. MaxPooling uygulandıktan sonraki boyutu ise 14*14*8*8 olmaktadır. Katmanlara girdikten sonra çıkan matrislerin boyutları Python'da *model.summary()* fonksiyonuyla görülebilmektedir. Kullanılan ağda katmanlardan çıkan matris boyutları aşağıdaki fotoğrafta görülmektedir.

Layer (type)	Output Shape	Param #
conv3d_1 (Conv3D)	(None, 28, 28, 8, 8)	1808
max_pooling3d_1 (MaxPooling3	(None, 14, 14, 8, 8)	0
conv3d_2 (Conv3D)	(None, 10, 10, 6, 16)	9616
max_pooling3d_2 (MaxPooling3	(None, 5, 5, 3, 16)	0
flatten_1 (Flatten)	(None, 1200)	0
dense_1 (Dense)	(None, 32)	38432
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 16)	528
dropout_2 (Dropout)	(None, 16)	0
dense_3 (Dense)	(None, 1)	17

Üstte anlatılan mantıkla diğer katmanların kodlanması yapılmıştır. CNN katmanlarından sonra Flatten katmanı gelmektedir. Üstteki fotoğrafta da görüldüğü üzere Flatten katmanında tek boyutlu 1200 elemanlı bir dizi oluşmaktadır. CNN katmanlarından çıkan dizinin boyutları *flattenArray()* fonksiyonuna verilip bu dizinin tek boyuta indirgenmesi sağlanmıştr. Sonrasında bu dizi Yapay Sinir Ağına verilecektir.

```
float* flattenArray = flatten3d(max_result_2, 5, 5, 3, 16);
```

Yapay Sinir Ağına verilmeden önce bu ağın ağırlıkların elde edilmesi gerekmektedir. Bu amaçla oluşturulan Weights sınıfı kullanılarak her katmanın ağırlıkları elde edilmiştir.

```
Weights w1(32, 38400);
w1.setLayerBias("conv3d_fc_bias_1.txt");
w1.setLayerWeight("conv3d_fc_kernel_1.txt");

Weights w2(16, 512);
w2.setLayerBias("conv3d_fc_bias_2.txt");
w2.setLayerWeight("conv3d_fc_kernel_2.txt");

Weights w3(1, 17);
w3.setLayerBias("conv3d_fc_bias_3.txt");
w3.setLayerWeight("conv3d_fc_kernel_3.txt");
```

Üstteki kod satırlarında w1 nesnesi birinci, w2 nesnesi ikinci, w3 nesnesi ise üçüncü katmanın ağırlıklarını tutmaktadır. İçine verilen parametreler ile ağırlıkları ait olduğu txt dosyalarından okumaktadır. Bu katmanların ağırlık sayıları da üstteki fotoğrafta görülebilir. Fotoğraftaki sayılar bias + ağırlık katsayıları şeklinde toplamı ifade etmektedir. C++'ta kodlamak için bu bias ve ağırlık katsayıları ayrı txt dosyalarında saklanmaktadır.

Ağırlıklar elde edildikten sonra kurulan ağın kodlanması gerçekleştirilmiştir.

```
#define FIRSTLAYERSIZE 32
#define FLATTEN_ARRAY_SIZE 1200

Neuron firstLayerNeuron;

    float firstTemp[FIRSTLAYERSIZE] = { 0 };
    int counter = 0, index;
    //FIRST NEURAL LAYER
    for (int id = 0; id < FIRSTLAYERSIZE; id++)
    {
        counter = id;
        index = 0;
        firstLayerNeuron.setId(id);
        for (int i = 0; i < FLATTEN_ARRAY_SIZE; i++)
        {
            firstTemp[id] = firstTemp[id] + (flattenArray[i] *
                                    w1.getLayerWeight((counter + index * FIRSTLAYERSIZE)));
             index++;
        }
        firstTemp[id] += w1.getLayerBias(id);
        firstLayerNeuron.setValue(firstTemp[id], id);
    }
```

Nöron değerlerini tutmak için önceden oluşturulan Neuron sınıfından yararlanılmıştır. İlk nörona ait ağırlıklar, ilk ağırlık alındıktan sonra katmanın nöron sayısı kadar atlanarak bulunmaktadır. Bu örnekte ilk katmanın nöron sayısı 32'dir. İlk katmanın ağırlıkları txt

dosyasının satırları göz önüne alınarak 0,32,64,96,... şeklinde gitmektedir. İkinci nöron ağırlıkları ise 1,33,65,… şeklinde gitmektedir. Bu mantıkla bir kodlama yapılmıştır. Üstte verilen kodda bu atlamaları yapmak amaçlı *index* değişkeni kullanılmıştır. Her katmanın nöron sayısı kadar atlayarak ağırlık değerlerini alıp ona gore işlemler yapıp nöron değerlerini hesaplamaktadır. Diğer iki katman için de aynı mantıkta kodlama gerçekleştirilmiştir. Bu şekilde sonuç nöronu hesaplanmıştır.

Bu hesaplamalar ardından kodun paralelleştirilmesi OpenMP Kütüphanesi yardımıyla sağlanmıştır. Bu şekilde daha hızlı bir şekilde çalışması sağlanmıştır.