Derin Öğrenme ile Modülasyonların Sınıflandırılması projesini gerçekleştirmek üzere bilgisayara öncelikle python 3.8 kurulumunu yapıp, kodların yazılacağı Anaconda platformu kuruldu.

Derin öğrenme kütüphanesi olarak Tensorflow kullanmaya karar verilip, Anaconda ortamında python 3.8 ile uyumlu olarak çalışabilen Tensorflow 2.5.0 kütüphanesi kuruldu. Ardından verilerin okunması, saklanması gibi işlemleri yapabilmek adına pandas kütüphanesi, verilerin işlenmesi, matematiksel ve array olarak işlem yapılabilmesi adına numpy kütüphanesi, verinin ve yapılan tüm işlemlerin görselleştirilebilmesi adına matplotlib kütüphanesi Anaconda platformuna kuruldu. Kod işlemleri hücresel yapısının kolaylık sağlamasından dolayı Anaconda Platformunun Jupyter Notebook ortamında yazılmaya karar verildi.


Derin öğrenmede, verileri eğitmek için kurulan modellerin katmanlarının fazla ve yapay sinir ağlarının çok fazla tensörden oluşan yoğun hesaplamalı işlemler gerçekleştirdiğinden ve kullanılacak dataların da boyutunun yüksek olmasından ötürü hesaplamalar direkt CPU ile yapılırsa işlem hacmi düşük olacaktır dolayısıyla dataların eğitilme aşamasında çok fazla vakit harcanacaktır. Bunun önüne geçmek adına bilgisayarın GPU'su ile veri eğitim işlemleri yapılması gerekmektedir. Gpu eş zamanlı, paralel hesaplama yapabilme özelliği ile optimizasyon, öğrenme işlemlerini çok daha hızlı gerçekleştirebilmektedir.

Kullanılan bilgisayarın,

CPU'su ( Intel i7.11 nesil), 8 çekirdekten oluşmaktadır.

GPU'su ise (MX450), 896 çekirdekli bir yapıya sahiptir.

Tensorflow kütüphanesini Nvidia GPU ile kullanabilmek için de Nvidia'nın sunmuş olduğu Cuda & Cudann yazılımlarının uygun sürümünlerini bilgasayara indirilmesi gerekmektedir.

GPU

| Sürüm | Python sürümü | Derleyici | Yapı araçları | cuDNN | CUDA |
|---|---|---|---|---|---|
| tensorflow-2.5.0 | 3.6-3.9 | KİK 7.3.1 | Bazel 3.7.2 | 8.1 | 11.2 |

Tensorflow 2.5 sürümüne konfigüre olan Cuda ve Cudann yazılımlarının uygun sürümlerini tespit ettikten sırasıyla Microsoft Visual Studio 2019, Cuda 8.1 ve Cudann 11.2.1 yazılımları indirilip yüklendi. Ayrıca Cudanın aktif bir şekilde çalışabilmesi için "bin" dosyasının çevre değişkenine path olarak eklenmesi gerekmektedir. Aşağıdaki verilen kod parçası gpu'nun aktif olduğunu göstermektedir.

```
In [1]: import tensorflow as tf
        import os

        print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

        Num GPUs Available:  1
```

**Radio ML 2016.a Dataset**

Bu projeyi gerçekleştirirken referans olarak kullanılan "Robust and Fast Automatic Modulation Classification with CNN under Multipath Fading Channels" makalesinde de kullanılmış olan RadioML 2016.a veri seti kullanılmaya karar verilmiştir.

RadioML 2016.a, değişken sinyal-gürültü oranlarında 11 modülasyondan (8 dijital ve 3 analog) oluşan, GNU Radio ile oluşturulmuş sentetik bir veri setidir.

- Modulasyon Türleri:
  - 8PSK
  - AM-DSB
  - AM-SSB
  - BPSK
  - CPFSK
  - GFSK
  - PAM4
  - QAM16
  - QAM64
  - QPSK
  - WBFM

Buradaki her bir modülasyon 20000 adet veriden oluşmaktadır, her bir modülasyonda -20db ile 18db arasında 2db aralıklarla 20 farklı SNR seviyesi bulunmaktadır. Her SNR seviyesi içinde 1000 adet veri bulunmaktadır.
Datasetin şekli (220000, 2, 128 )'dir. Totalde 220000 adet veri vardır. Her bir veri de real ve imajiner eksen olmak üzere iki eksen olarak 2x128lik bilgi barındırmaktır.

RadioML 2016.a Veri setinin okunması ve preprocessing işlemleri gerçekleştirildi.

```
In [3]: path = "C:/Users/AhmetFurkan/Desktop/RML2016.10a_dict.pkl"

        df = pd.read_pickle(path)
```

```
In [4]: key_arr = list(df.keys())   # tüm modülasyon tiplerinin tüm snr seviyeleri (220 adet) key_arr'e atandı
        classes = []

        for i in range(len(key_arr)):    # Sınıflar classes list'e aktarıldı.
            temp_key = key_arr[i][0]
            if temp_key not in classes:
                classes.append(temp_key)

        classes = np.array(classes)     #  classes list'i array'e dönüştürüldü.
        classes = classes.reshape(classes.shape[0], 1)
```

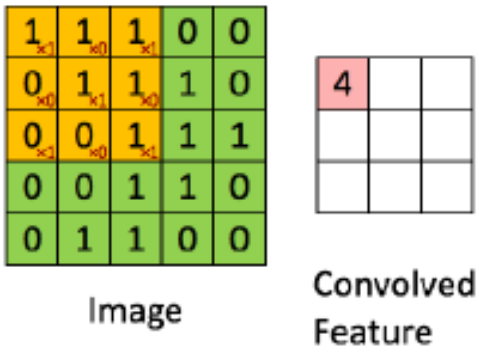
```
In [18]: classes

Out[18]: array([['QPSK'],
                ['PAM4'],
                ['AM-DSB'],
                ['GFSK'],
                ['QAM64'],
                ['AM-SSB'],
                ['8PSK'],
                ['QAM16'],
                ['WBFM'],
                ['CPFSK'],
                ['BPSK']], dtype='<U6')
```

Okunan veri setinin sınıflarını classes adı altında array'e atama yaparak sınıflar belirlendi.

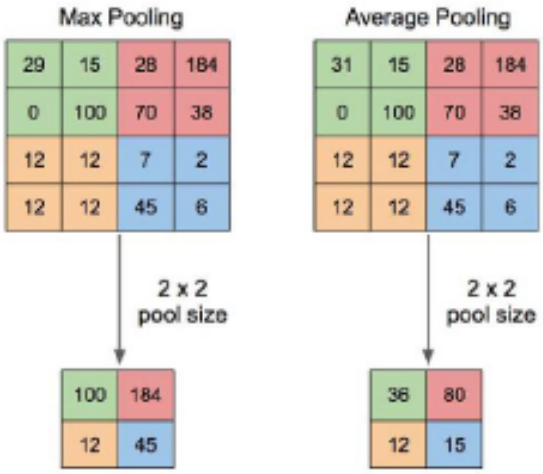
Ardından X ve lbl arrayleri oluşturularak X'e veri setininin kendisi lbl ise etiketler atanarak veri önişleme aşaması tamamlanmış oldu.

Son olarak dataseti train ve test olarak ayırma işlemi gerçekleştirildi. Verisetinin %20'si train_test_split metoduyla test veriseti olarak ayrıldı. Ayrıca data üzerindeki kategorik verilerin temsilinin daha etkileyici ve kolay olmasını sağlamasından ötürü OneHotEncoding() yöntemi kullanıldı. One Hot Encoding'de veriler binary şekilde ( mevcut değere 1 verip mevcut olmayan değere 0 verilir ) ifade edilir.

```
In [19]: X_train, X_test, y_train, y_test = train_test_split(X, lbl, test_size = 0.2, random_state = 2000)
         enc = OneHotEncoder()
         enc.fit(classes)

         y_train_ = []
         y_train_snr = []

         for i in y_train:
             y_train_.append(i[0])
             y_train_snr.append(i[1])


         y_train = np.array(y_train)
         y_test = np.array(y_test)
         y_train = y_train.reshape(y_train.shape[0], 1)
         y_test = y_test.reshape(y_test.shape[0], 1)
         y_train = enc.transform(y_train).toarray()
         y_test = enc.transform(y_test).toarray()

         #Reshape

         #X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
         #X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)
```

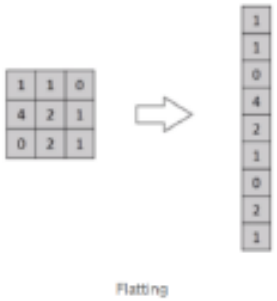



Conv2D ( Konvolüsyon) katmanı, CNN modellerin temelini oluşturan olmazsa olmaz katmandır. Filtre sayısı input ile konvole olacak çekirdek sayısını belirler. Filtre sayısını ne kadar arttırırsak işlem hacmi o kadar artacaktır. Kernel size ise bir matris oluşturarak veri üzerinde adım adım bir çarpım işlemine tabi tutar.




MaxPooling ( Havuzlama) diğer bir adıyla downsampling katmanı, belirli bölgedeki tüm değerleri birbiriyle karşılaştırır ve maksimum piksel değeri dikkate alınır. Bu katmanın görevi, gösterimin kayma boyutunu ve ağ içindeki parametreleri ve hesaplama sayısını azaltmak içindir. Bu sayede ağdaki uyumsuzluk kontrol edilmiş olur ve nöronların görüntünün belirli bir yamasından daha fazla özellik çıkarmasına yardımcı olur.

Dropout ise gizli nöronları, girilen parametre sayısı olasılıkla 0'a ayarlar. Böyle yaparak modelin genelleştirilmesine yardımcı olur. Modelin overfitting (aşırı öğrenme) olması engellenir.




Flatten katmanının görevi basitçe, son ve en önemli katman olan Fully Connected Layer'ın (Dense) girişindeki verileri hazırlamaktır. Genel olarak, sinir ağları, giriş verilerini tek boyutlu bir diziden alır. Bu sinir ağındaki veriler ise Convolutional ve Pooling katmanından gelen matrixlerin tek boyutlu diziye çevrilmiş halidir.

Çıkış katmanı olarak sınıflamada kullanılan standart sinir ağı olan Dense Layer kullanılmaktadır. Bu katmanda, girdideki her bir düğüm çıkıştaki her bir düğüm ile bağlıdır. Çıkarılan nitelikleri istenen çıktıya eşleyen son öğrenme aşaması olarak görülebilir.

Loss Function Ve Optimizer

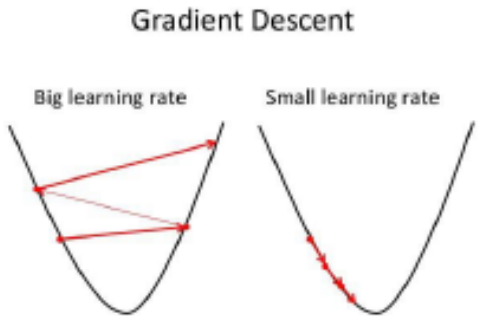
Sistem, yaptığı tahminleri için hata kontrolü yapar ve hatayı sürekli olarak minimize etmeye çalışır. Bunun için hatayı hesaplayıp(loss function) beklenen sonuca yakınsamaya(optimizer kullanır) yani hatayı azaltmaya çalışır. Loss function ile hata hesaplanırken optimizer(örneğin: gradient descent) ile en düşük hata alınacak ihtimaller bulunulmaya çalışılır. Hatanın minimize edilmesi için kullanılan bu fonksiyonlara optimizer denir.

## Optimizer

Some optimizer varieties;

- Adam
- SGD
- RMSprop
- Adadelta
- Adagrad
- Adamax
- Ftrl
- Nadam

Optimizasyonun önemli parametrelerinden biri olan, Learning Rate, bir ağın parametrelerini ne kadar hızlı güncellediğini tanımlar. Düşük öğrenme oranı, öğrenme sürecini yavaşlatır ancak sorunsuz bir şekilde birleşir. Daha büyük öğrenme oranı, öğrenmeyi hızlandırır, ancak yakınsama olmayabilir.




Epoch Sayısı, eğitim sırasında tüm eğitim verilerinin ağa gösterilme sayısıdır. Eğitimin doğruluğu arttığında bile, (validation) doğruluğun en yüksek olduğu yerde epoch sayısı idealdir.

Batch size parametresi olarak belirlenen değer; modelin aynı anda kaç veriyi işleyeceği anlamına gelir.

EarlyStopping fonksiyonunun içinde yer alan Patience değeri validation error test error den daha büyük değerler aldığında kaç epoch daha yapabileceğini belirleyen sayıdır.

```
opt = tf.keras.optimizers.Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.99, amsgrad = False)
callback = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', patience = 5)

model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
                    callbacks = [callback],
                    epochs=200, batch_size = 16)

# lr 0001 beta 025 beta2 05 amsgard false
```

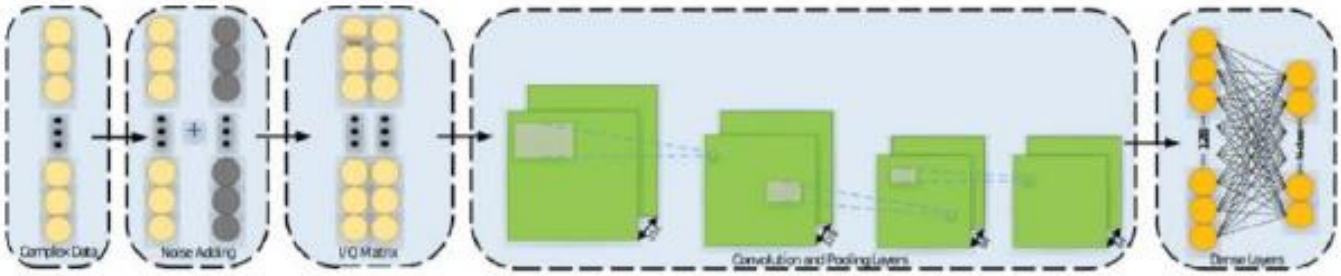



Fig. 1. The proposed CNN model consists of four convolution and pooling layers and two dense layers.