



**EGE ÜNİVERSİTESİ  
MÜHENDİSLİK FAKÜLTESİ  
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ  
2020 – 2021  
GÜZ DÖNEMİ  
GÖRÜNTÜ İŞLEME DERSİ**

**PROJE 2  
DERİN ÖĞRENME İLE GÖRÜNTÜ TANIMA  
/ NESNE TESPİTİ**

Ege Doğan Dursun 05170000006

## **1. Image Classification, Object Detection ve Image Segmentation kavramları arasındaki farklar nelerdir? Belirtiniz. (5)**

Image Classification, görselleri çeşitli paternlere ve bağlantılaraya dayalı olarak sınıflandırmaya verilen isimdir. Makine öğrenmesi söz konusu olduğunda, bu tür sınıflandırmalarda görsellerin arasındaki farkları ve özgünlüklerini belirten çeşitli paternler ve öznitelikler tanımlanarak bu sayede çeşitli görsellerin sınıflandırılması ve etiketlenmesi gerçekleştirilir. Object Detection, çeşitli tanımlara ve sınıflara bağlı olan objelerin görsel bir uzay içerisinde tespit edilmesini ifade eden bir kavramdır. Yine bu kavramın içerisinde de çeşitli objeler öznitelikleri ile ayırt edilirler. Image Segmentation ise görüntünün bölütlenmesini ifade etmek için kullanılmaktadır. Bu fonksiyon, kimi zaman görüntüdeki belli detayların ön plana çıkarılması için, kimi zaman ise görüntüdeki ana temanın daha da güçlendirilmesi için kullanılabilmekte ve özellikle tipta çeşitli dokuların tespiti ve ayrıştırılması gibi konularda yoğun olarak kullanılmaktadır. [1][2][3]

Image Classification, Object Detection fonksiyonu için gerekli olan bir kavramdır. Çünkü bir objeyi tespit etmek istiyorsak onu öncelikle tanımlayabilmeliyiz. Buna ek olarak bir görüntüdeki belirli öğelerin diğerlerinden öne çıkması da onun ayrıştırılmasını kolaylaştırmaktadır. Bu durumda Image Segmentation aslında Image Classification ve dolayısıyla Object Detection konusunda da anahtar bir kavram olabilmektedir.

**2. Ön Çalışma: Ders notlarından derin öğrenme (deep learning) ile ilgili sunum ve ders videosunu inceledikten sonra internetten derin öğrenme ile ilgili görüntü tanıma örnekleri araştırarak yazılımları okuyunuz. İlgili konularda anlatım yapılan videolar izleyiniz. İncelediğiniz örneklerin, izlediğiniz video ve diğer kaynakların linkeri ile, araştırmanızdan kısaca neler öğrendiğinizi rapora ekleyiniz. (5)**

Proje dahilinde öncelikli olarak keras kütüphanesine ait modüllerin dökümentasyonunu inceledim ve kullanılan metodların hangi amaçlar ile entegre edilebildiğini anlamaya çalıştım. [4]

Ardından, keras kullanılarak oluşturulan örnek Evrişimsel Sinir Ağı modellerine yönelik bir araştırma yaptım ve bu örnek modelleri kendim oluşturmak üzerine uğraştım. [5][6] Geliştirdiğim modellerin yine keras, sklearn, matplotlib gibi kütüphaneler kullanılarak nasıl performans ölçümlemesine tabi tutulabileceği ve elde edilen verilerin nasıl görselleştirileceği konusunda araştırmalar gerçekleştirdim. Daha etkin görselleştirmeler yapabilmek için Matplotlib kütüphanesine ait dökümentasyon üzerine çalıştım. [7]

Evrişimsel sinir ağlarındaki Evrişim ve Havuz katmanlarının nasıl çalıştığını ve arkalarındaki mantığı öğrenmek için çeşitli sitelerden araştırmalar yaptım. Düzleştirme, Dropout, Adam optimizasyonu, aktivasyon fonksiyonları gibi çeşitli konularda bilgi almak için çeşitli kaynakları araştırdım. [8][9][10]

Öğrendiklerimi devam eden cümlelerle özetleyebilirim.

Öncelikli olarak Evrişimsel Sinir Ağları aslında beynimizin görsel korteksindeki yorumlayıcı hücrelerin ve retinamızdaki ışık algılayıcı hücrelerin birlikte gerçekleştirdiği fonksiyona çok yakın bir işlev sağlıyor. Konvolüsyon ve Havuz katmanları sayesinde görseller tek bir büyük parça yerine küçük parçalara ayırtırılarak ve çeşitli filtreler uygulanarak farklı açılardan inceleniyor ve bu tıpkı beynimizde olduğu gibi işlem tasarrufu için veri sıkıştırma yöntemlerine tabi tutuluyor. Aktivasyon fonksiyonlarını da beynimizdeki nöronların eylem potansiyeline benzetebiliriz. Elbette beyindekinden farklı olarak yazılımsal ortamda ifade edilen nöronlar kimyasal değil, matematiksel olarak çalışmaktadır. Bu noktada çeşitli aktivasyon fonksiyonları mevcut. Bunlardan günümüzde en çok tercih edileni ve en performanslı olduğu iddia edileni ReLU ismindeki aktivasyon fonksiyonu. Elbette, çeşitli veri bilimi problemleri için belirli başka aktivasyon fonksiyonları özel olarak daha verimli olabiliyor.

Son olarak karşılaştığım ilginç detaylardan biri, örneğin tıbbi görüntüler gibi çok büyük boyutlu görsellerde, kimi zaman yapay sinir ağının çok küçük ayrıntıları tespit etmede başarısız olmasışıydı. Bunu engellemek için görüntüyü tek bir büyük görsel yerine küçük patch'lere ayırarak beyindeki piramidal nöronlara benzeten Residual Sinir Ağları kullanarak işlemek çok daha verimli sonuç veriyormuş. Aynı şekilde, kullanılan görsellerde hangi noktanın problematik olduğunun maskelenme yöntemi ile işaretlenmesi de verimin arttırılmasını kolaylaştırın etkenlerden olarak belirtiliyor.

### **3. Derin Öğrenme ile Görüntü Tanıma / Nesne Tespit Yazılımı Geliştirme**

#### **3.1 Konu Belirleme ve Problemin Tanımı (5)**

Projemde keras kütüphanesinin içерdiği veri setleri arasında yer alan MNIST veri setini kullanarak bir Evrişimsel Sinir Ağı (CNN) tabanlı görüntü tanıma yazılımı geliştirmeye çalıştım. Geliştirme sürecinin ardından da, hiper parametreleri optimize etmek üzerine uğraşarak geliştirdiğim modelin doğruluk oranlarını olabildiğince yükseltmek için çaba sarfettim.

MNIST veri seti el yazısı ile çizilmiş rakamlardan meydana gelmektedir ve dahilinde 60.000 adet eğitim görseli ve 10.000 adet test görseli bulundurmaktadır. Görsellerin boyutu 28x28 ebatlarındadır ve siyah beyazdır. MNIST veri seti dahilindeki görsellerin birçoğu NIST verisetine ait olmakla birlikte, rakamlara ek olarak karakterleri de içeren ve 240.000 eğitim, 40.000 test görselinden oluşan EMNIST adında bir veri seti de bulunmaktadır.

#### **3.2 Veriseti Hakkında Bilgi (Kullanılan / Oluşturulan) (5)**

Projede kullanılan veri seti keras kütüphanesi dahilinde de hazır olarak kullanılabilen MNIST veri setidir. MNIST veri seti 60.000 adet eğitim, 10.000 adet test görselinden oluşan ve el yazısıyla yazılmış rakam görselleri içeren bir veri setidir. Bu veri setindeki görsellerin ebatları 28x28'dir ve siyah beyaz görsellerdir.

10'luk sayı sisteminde toplam 10 adet rakam bulunduğuundan, MNIST veri setindeki toplam sınıf sayısı da 10 adettir. Bu rakamlar 0, 1, 2, 3, 4, 5, 6, 7, 8 ve 9'dur.

MNIST veri setine ait görseller Amerikalı Census Bureau çalışanları ve Amerikalı lise öğrencilerinin el yazılarından meydana gelmektedir. 1999 yılında NYU'dan Yann LeCun tarafından oluşturulmuştur.

Veri seti 1999 yılından beri çeşitli makine öğrenmesi ve derin öğrenme modellerinin test edilmesinde kullanılmaktadır ve neredeyse standart denilebilecek bir ölçek haline gelmiştir. Kimi zaman derin öğrenme üzerine özelleştirilmiş belirli cihazların performansları kıyaslanırken MNIST veri seti kullanılarak örneğin ResNet50 gibi hazır sinir ağları üzerinde ölçümlenme yapılır ve sonuçlar değerlendirilir.

### **3.3.1 Kullandığınız ortam, dil, kütüphane adı ve sürümlerini yazınız. (5)**

Projemde kullandığım ortam Python 3 uyumlu bir IDE olan Spyder'dı. Spyder'ın 3.6.6 sürümü üzerinde çalışmalarımı gerçekleştirdim. Kullandığım Python sürümü ise Python 3.7.0'dı. Projem dahilinde derin öğrenme konusunda fayda sağlayan çeşitli kütüphanelere ait fonksiyonları kullandım. Bunlara örnek olarak keras, numpy, matplotlib, seaborn ve sklearn gösterilebilir. Kullandığım bu kütüphanelerin versiyonları aşağıdaki gibidir.

- **Keras** : 2.4.3
- **Numpy** : 1.19.4
- **Matplotlib** : 3.1.2
- **Seaborn** : 0.11.0
- **Sklearn** : 0.24.0

**3.3.2 Derin Öğrenme ile bir tanıma / tespit uygulaması geliştiriniz, modeli oluşturunuz, model adı ve katman sayısı bilgilerini, kodu ve modelin şemasını rapora ekleyiniz. (10)**

Projemde kullandığım derin öğrenme modeli bir Evrişimsel sinir ağı mimarisidir. Modelin tanımı proje dahilinde “model.py” isminde ayrı bir dökümda gerçekleştirılmıştır.

Modelde Sequential mimari tercih edilmiştir ve ilgili mimariyi oluşturmak için keras.models modülünden faydalanyılmıştır. Modelde Evrişim işlemlerini gerçekleştirmek için keras.layers modülüne ait Conv2D ve MaxPooling2D sınıflarından yararlanılmıştır. BatchNormalization ve Dropout katmanları ile performans optimizasyonu sağlanması hedeflenmiştir. Buna ek olarak evrişim çıktılarını düzleştirmek için Flatten sınıfı ve yoğun katmanlar için de Dense sınıfı kullanılmıştır.

Aşağıdaki görsellerde modele yönelik mimarinin Python kodu şeklindeki ifadesini inceleyebilirsiniz.

```
def get_model(input_shape):  
  
    model = Sequential()  
  
    model.add(  
        Conv2D(  
            filters=32,  
            kernel_size=3,  
            activation='relu',  
            input_shape=input_shape,  
        )  
    )  
  
    model.add(  
        BatchNormalization()  
    )  
  
    model.add(  
        Conv2D(  
            filters=32,  
            kernel_size=3,  
            activation='relu',  
        )  
    )  
  
    model.add(  
        BatchNormalization()  
    )
```

**Görsel 1:** Modeli tanımlamak ve çağırabilmek için yaratılan *model.py* sınıfındaki koda ait görsüntü.

```
52  
53     model.add(  
54         Conv2D(  
55             filters=32,  
56             kernel_size=5,  
57             strides=2,  
58             padding='same',  
59             activation='relu',  
60             )  
61     )  
62  
63     model.add(  
64         BatchNormalization()  
65     )  
66  
67     model.add(  
68         Dropout(0.4)  
69     )  
70  
71     model.add(  
72         Conv2D(  
73             filters=64,  
74             kernel_size=3,  
75             activation='relu',  
76             )  
77     )  
78  
79     model.add(  
80         BatchNormalization()  
81     )  
82  
83
```

Görsel 2: Modeli tanımlamak ve çağrılabilmek için yaratılan *model.py* sınıfındaki koda ait görüntüsü.

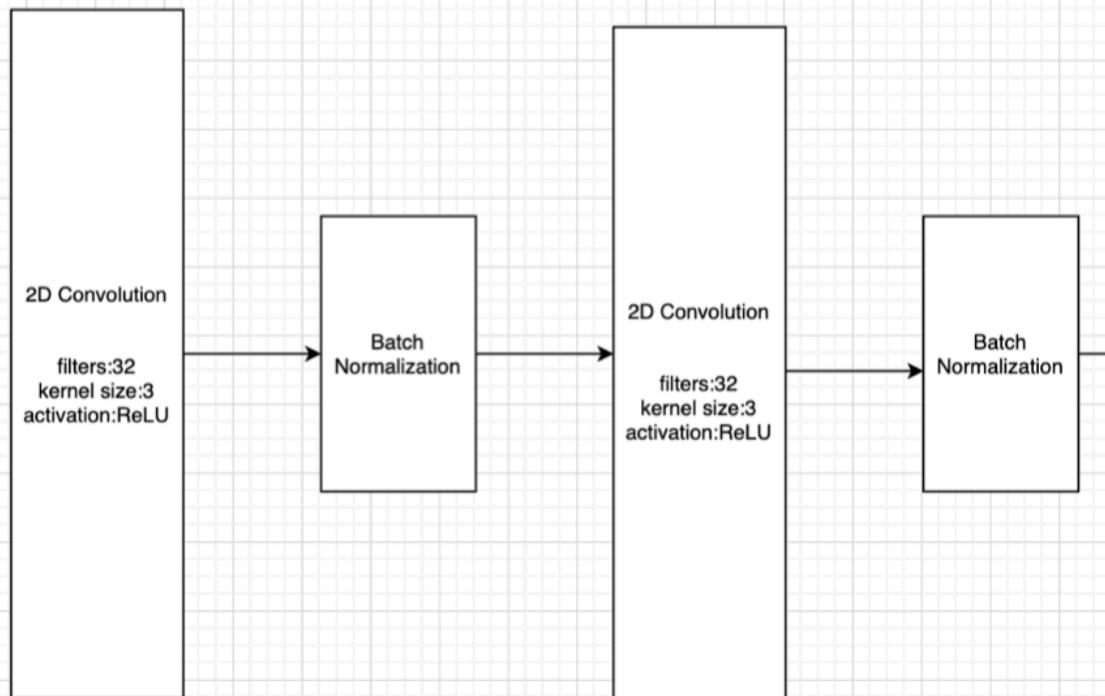
```
83
84     model.add(
85         Conv2D(
86             filters=64,
87             kernel_size=3,
88             activation='relu',
89             )
90         )
91
92     model.add(
93         BatchNormalization()
94         )
95
96
97     model.add(
98         Conv2D(
99             filters=64,
100            kernel_size=5,
101            strides=2,
102            padding ='same',
103            activation='relu',
104            )
105        )
106
107    model.add(
108        BatchNormalization()
109        )
110
```

**Görsel 3:** Modeli tanımlamak ve çağrılabilmek için yaratılan *model.py* sınıfındaki koda ait görüntüsü.

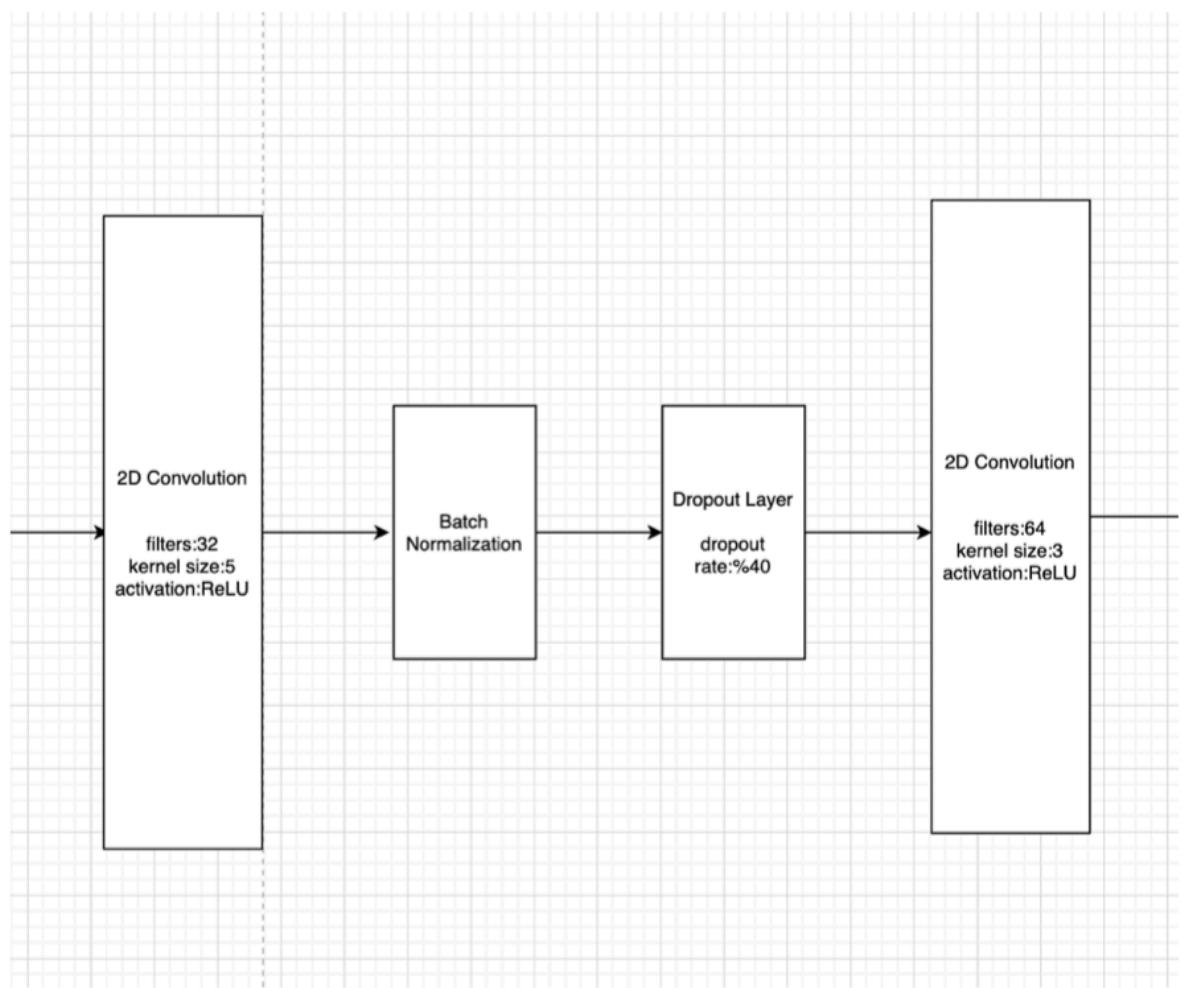
```
111     model.add(  
112         Dropout(0.4)  
113     )  
114  
115     model.add(  
116         Flatten()  
117     )  
118  
119     model.add(  
120         Dense(  
121             128,  
122             activation='relu',  
123             )  
124         )  
125  
126     model.add(  
127         BatchNormalization()  
128     )  
129  
130     model.add(  
131         Dropout(0.4)  
132     )  
133  
134     model.add(  
135         Dense(  
136             10,  
137             activation='softmax',  
138             )  
139         )  
140  
141  
142     model.compile(  
143         optimizer='adam',  
144         loss='categorical_crossentropy',  
145         metrics=["accuracy"],  
146         )  
147  
148     return model
```

Görsel 4: Modeli tanımlamak ve çağrılabilmek için yaratılan model.py sınıfındaki koda ait görüntüsü.

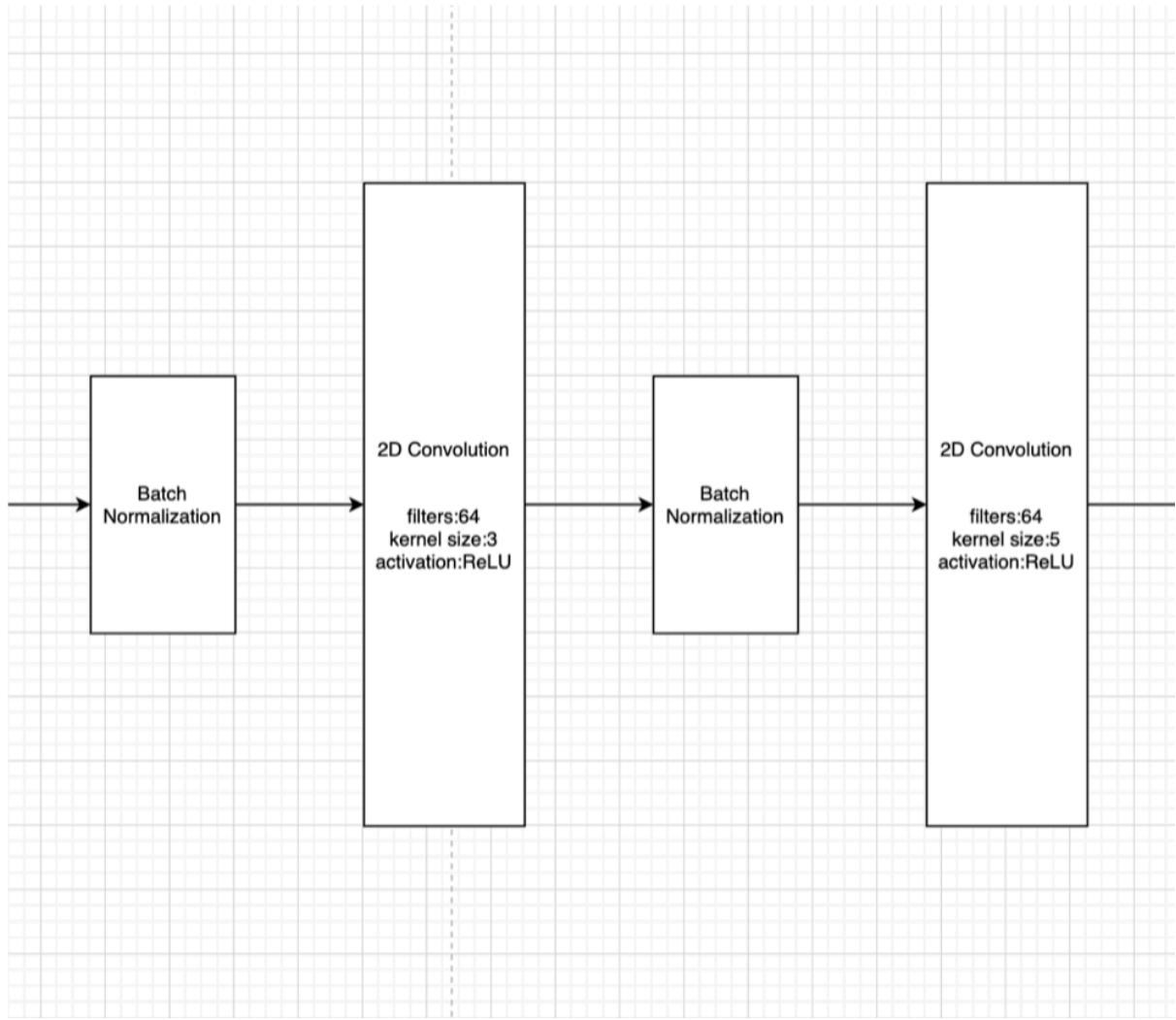
Bu görselere ek olarak modelin mimarisini ifade eden aşağıdaki çizgeleri de inceleyebilirsiniz.



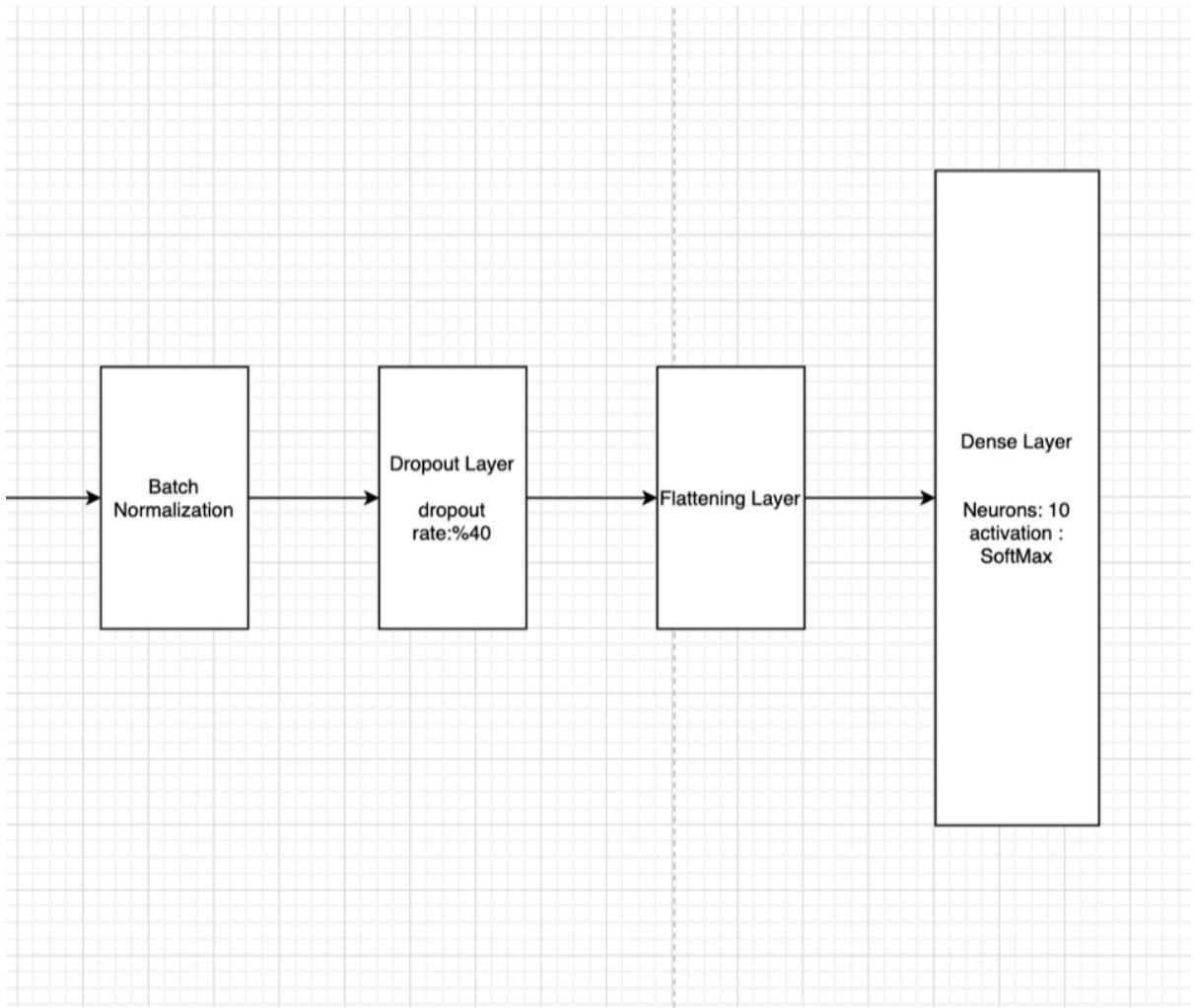
Görsel 5: Model Şeması / Parça 1



**Görsel 6:** Model Şeması / Parça 2



Görsel 7: Model Şeması / Parça 3



Görsel 8: Model Şeması / Parça 4

Modelin katmanlarına ve bu katmanların özelliklerine ait bilgileri de aşağıda inceleyebilirsiniz.

1. **Evrişim Katmanı** -> Filtre: 32, Kernel Boyutu:3, Aktivasyon: ReLU
2. **Batch Normalizasyonu Katmanı**
3. **Evrişim Katmanı** -> Filtre: 32, Kernel Boyutu: 3, Aktivasyon: ReLU
4. **Batch Normalizasyonu Katmanı**
5. **Evrişim Katmanı** -> Filtre: 32, Kernel Boyutu: 5, Stride: 2 Aktivasyon: ReLU
6. **Batch Normalizasyonu Katmanı**
7. **Dropout Katmanı** -> Dropout Oranı: %40
8. **Evrişim Katmanı** -> Filtre: 64, Kernel Boyutu: 3, Aktivasyon: ReLU
9. **Batch Normalizasyonu Katmanı**
10. **Evrişim Katmanı** -> Filtre: 64, Kernel Boyutu: 5, Stride: 2, Aktivasyon: ReLU
11. **Batch Normalizasyonu Katmanı**
12. **Evrişim Katmanı** -> Filtre: 64, Kernel Boyutu: 5, Stride: 2, Aktivasyon: ReLU
13. **Batch Normalizasyonu Katmanı**
14. **Dropout Katmanı** -> Dropout Oranı: %40
15. **Düzleştirme Katmanı**
16. **Yoğun Katman** -> Nöron Sayısı: 128, Aktivasyon: ReLU
17. **Batch Normalizasyonu Katmanı**
18. **Dropout Katmanı** -> Dropout Oranı: %40
19. **Yoğun Katman** -> Nöron Sayısı: 10, Aktivasyon: SoftMax

Yukarıda belirtilen katman bilgilerine bağlı olarak geliştirdiğim sinir ağı modelinde:

- 19 adet katman bulunmaktadır.
- 6 adet evrişimsel katman bulunmaktadır.
- İlk 3 evrişimsel katmanda 32 filtre kullanılmıştır.
- Son 3 evrişimsel katmanda 64 filtre kullanılmıştır.
- Optimizasyon yöntemi olarak Adam Optimizasyonu kullanılmıştır.
- Loss fonksiyonu olarak Categorical Crossentropy kullanılmıştır.

Modele ait gerekli tanımları ve gerçekleştirdikten sonra eğitim sürecinin başlatılması için gerekli kodun yazılması gereklilik haline geldi. Bu amaçla modelin eğitimi için kullanılacak olan “training.py” isminde bir Python dökümanı oluşturdum.

Bu doküman dahilinde öncelikli olarak keras, numpy vb. gerekli üçüncü parti kütüphaneleri import ettim ve elbette modele ait mimariyi çağırabilmek için “model.py” modülünü import ettim.

Sonrasında batch boyutu, sınıf sayısı ve eğitimin gerçekleştirileceği epoch sayısı gibi belirli hiperparametreleri tanımladım. MNIST veri setini keras.datasets modülünü kullanarak eğitim ve test setleri ayrı olacak şekilde çağrırdım. Theano ve Tensorflow'un görüntü kanallarını ifade ediş formatları birbirinden farklı olduğu için karışıklığı önlemek amacıyla güvence sağlayan ufak bir kod yazdım.

Sonrasında görüntülerin normalizasyonu ve ölçeklendirilmesi için kanal değerlerini 0 ve 1 arasına denk gelecek şekilde ayarladım. Sonrasında etiket değerlerini kategorik değerlere dönüştürdüm.

Önceki görsellerde model.py sınıfında tanımının gerçekleştirildiği modelimi ilgili get\_model metodunu kullanarak çağrırdım. Oluşturduğum modelin h5 formatında kaydedilebilmesi ve modelle ilgili gelişmelerin eğitim süresince kaybolmaması için ModelCheckpoint sınıfından faydalandım.

Ardından eğitim verilerini, belirlenen epoch sayısı boyunca eğitmek üzere modele entegre ettim ve eğitimin tamamlanmasını bekledim.

Modelin oluşturduğu Loss ve Accuracy değerlerini kullanıcıya görüntülemenin ardından örnek tahminlemeler gerçekleştirdim. Sonrasında Modelin eğitim sürecine ait Loss ve Accuracy değerlerinin zaman içerisindeki değişimlerini grafikler halinde gösterdim. Son olarak tahminlenen örneklerin doğruluklarını bir Confusion Matrix üzerinde göstererek kullanıcıya görselleştirdim.

Training.py isimli Python model eğitim dökümanına ait kod görselleri aşağıda belirtilmiştir.

```

19 #Import the necessary libraries
20 import keras
21 from keras.datasets import mnist
22 from keras import backend as K
23 from model import get_model
24 import numpy as np
25 from keras.callbacks import ModelCheckpoint
26 import matplotlib.pyplot as plt
27 import tensorflow as tf
28 import seaborn as sns
29
30 #Determine the batch size, number of classes and total epochs
31 batch_size = 32
32 num_classes = 10
33 epochs = 30
34
35 #height and width of the images
36 rows, cols = 28, 28
37
38
39 #Get the dataset
40 (x_train, y_train), (x_test, y_test) = mnist.load_data()
41
42
43 #Format the data for the channels
44 if K.image_data_format() == 'channels_first':
45     x_train = x_train.reshape(x_train.shape[0], 1, rows, cols)
46     x_test = x_test.reshape(x_test.shape[0], 1, rows, cols)
47     input_shape = (1, rows, cols)
48
49 else:
50     x_train = x_train.reshape(x_train.shape[0], rows, cols, 1)
51     x_test = x_test.reshape(x_test.shape[0], rows, cols, 1)
52     input_shape = (rows, cols, 1)
53

```

**Görsel 9:** *Training.py* dökümanına ait kodların görüntüsü

```

55 #Preprocess and normalize the data
56 x_train = x_train.astype('float32')
57 x_test = x_test.astype('float32')
58
59 x_train = x_train / 255
60 x_test = x_test / 255
61
62 y_train = keras.utils.to_categorical(y_train, num_classes)
63 y_test = keras.utils.to_categorical(y_test, num_classes)
64
65
66 #Get the model we will use for the training
67 model = get_model(input_shape)
68
69 #Determine a file path to save the model
70 filepath = "model.hdf5"
71
72 #Create checkpointing callback object for saving the model
73 checkpoint = ModelCheckpoint(filepath,
74                             monitor='val_accuracy',
75                             verbose=1,
76                             save_best_only=True,
77                             mode='max')
78
79
80 #Start training the model
81 history = model.fit(
82     x_train,
83     y_train,
84     batch_size = batch_size,
85     epochs = epochs,
86     verbose=1,
87     validation_data = (x_test, y_test),
88     callbacks = [checkpoint],
89 )
90

```

**Görsel 10:** *Training.py* dökümanına ait kodların görüntüüsü

```

91
92 #Evaluate the model
93 score = model.evaluate(x_test, y_test, verbose=1)
94
95
96 #Show model performance
97 print("_____")
98 print("Model Loss : ", score[0])
99 print("Model Accuracy : %", score[1]*100)
100 print("_____")
101
102
103 #Create predictions
104 predictions = model.predict(x_test)
105
106
107 #Plot the training accuracy history
108 plt.plot(history.history['accuracy'])
109 plt.title('Training Accuracy History')
110 plt.ylabel('Accuracy Value (%)')
111 plt.xlabel('Number of Epoch')
112 plt.show()
113
114
115 #Plot the validation accuracy history
116 plt.plot(history.history['val_accuracy'])
117 plt.title('Validation Accuracy History')
118 plt.ylabel('Accuracy Value (%)')
119 plt.xlabel('Number of Epoch')
120 plt.show()
121

```

**Görsel 11:** *Training.py* dökümanına ait kodların görselini

```

122
123 #Plot the training and validation accuracy history
124 plt.plot(history.history['accuracy'])
125 plt.plot(history.history['val_accuracy'])
126 plt.ylabel('Accuracy Value (%)')
127 plt.xlabel('Number of Epoch')
128 plt.show()
129
130
131 y_trues = []
132 for i in range(0, len(y_test)):
133     y_trues.append(np.argmax(y_test[i]))
134
135 y_preds = []
136 for j in range(0, len(predictions)):
137     y_preds.append(np.argmax(predictions[j]))
138
139
140 #Create the confusion matrix
141 con_mat = tf.math.confusion_matrix(
142     labels = y_trues,
143     predictions = y_preds,
144 ).numpy()
145
146
147 #Print the confusion matrix
148 print("\n HATA MATRİSİ : \n")
149 figure = plt.figure(figsize=(8, 8))
150 sns.heatmap(con_mat, annot=True,cmap=plt.cm.Blues, fmt='g')
151 plt.tight_layout()
152 plt.ylabel('True label')
153 plt.xlabel('Predicted label')
154 plt.show()
155
156

```

**Görsel 12:** *Training.py* dökümanına ait kodların  
görüntüsü

Modelin oluşturulması ve kaydedilmesinden sonra kaydedilen modelin sınanması için “*test\_model.py*” isminde ayrı bir döküman meydana getirilmiştir.

Bu doküman dahilinde MNIST veri setine ait sadece test verileri alınır ve bu veriler kullanılarak oluşturulan model üzerinde performans değerlendirmeşi gerçekleştirilir. Sonuç olarak elde edilen Loss ve Accuracy değerleri kullanıcıya görüntülenerek bir Confusion Matrix kullanılarak da ifade edilir.

Bahsedilen *test\_model.py* Python dökümanına ait kod görselleri aşağıda paylaşılmıştır.

```

19 #Import the necessary libraries
20 import keras
21 from keras.datasets import mnist
22 import matplotlib.pyplot as plt
23 from keras import backend as K
24 from keras.models import load_model
25 import os
26 import tensorflow as tf
27 import seaborn as sns
28 import numpy as np
29
30
31 #height and width of the images
32 rows, cols = 28, 28
33 num_classes = 10
34
35
36 #Get the dataset
37 (_, _), (x_test, y_test) = mnist.load_data()
38
39
40 #Format the data for the channels
41 if K.image_data_format() == 'channels_first':
42     x_test = x_test.reshape(x_test.shape[0], 1, rows, cols)
43     input_shape = (1, rows, cols)
44
45 else:
46     x_test = x_test.reshape(x_test.shape[0], rows, cols, 1)
47     input_shape = (rows, cols, 1)
48

```

**Görsel 13:** *test\_model.py* dosyamına ait kodların  
görüntüsü

```

48
49
50 #Preprocess and normalize the data
51 x_test = x_test.astype('float32')
52
53 x_test = x_test / 255
54
55 y_test = keras.utils.to_categorical(y_test, num_classes)
56
57
58 #Load the model
59 cwd = os.getcwd()
60 path = os.path.join(cwd, 'model.hdf5')
61 model = load_model(path)
62
63
64 #Evaluate the model
65 score = model.evaluate(x_test, y_test, verbose=1)
66
67
68 #Show model performance
69 print("_____")
70 print("Model Loss : ", score[0])
71 print("Model Accuracy : %", score[1]*100)
72 print("_____")
73
74
75 #Create predictions
76 predictions = model.predict(x_test)
77
78

```

**Görsel 14:** *test\_model.py* dökümanına ait kodların  
görüntüsü

```

79
80 y_trues =[]
81 for i in range(0, len(y_test)):
82     y_trues.append(np.argmax(y_test[i]))
83
84 y_preds = []
85 for j in range(0, len(predictions)):
86     y_preds.append(np.argmax(predictions[j]))
87
88
89 #Create the confusion matrix
90 con_mat = tf.math.confusion_matrix(
91     labels = y_trues,
92     predictions = y_preds,
93     ).numpy()
94
95
96 #Print the confusion matrix
97 print("\n HATA MATRİSİ : \n")
98 figure = plt.figure(figsize=(8, 8))
99 sns.heatmap(con_mat, annot=True,cmap=plt.cm.Blues, fmt='g')
100 plt.tight_layout()
101 plt.ylabel('True label')
102 plt.xlabel('Predicted label')
103 plt.show()
104
105

```

**Görsel 15:** *test\_model.py* dökümanına ait kodların görüntüüsü

**(Modelin optimize edilmesi ve K-Fold Cross Validation kullanılarak iyileştirilmesine yönelik çalışmalar 4 numaralı başlık altında anlatılmıştır.)**

### 3.3.3 Hangi çalışma / hazır kod / bağlantılardan faydalandınız? Faydalandığınız çalışma ve kaynaklara göre farklılıklarınız nelerdir? Açıklayınız. (10)

MNIST veri seti oldukça popüler bir veri seti olduğundan internette başlangıç düzeyinden, ileri düzeye kadar birçok eğitim koduna ulaşmak mümkün. Benzer çalışmalarla yönelik çeşitli araştırmalar yaptığımda bilinen en iyi MNIST tanımlayıcı modelinin ulaştığı doğruluk değerinin %99,8 olduğunu tespit ettim. Benim geliştirdiğim modelin ise ulaşabildiği maksimum doğruluk oranı yaklaşık %99,62. Kaggle üzerinden gerçekleştirilen çalışmalarda ise ulaşılabilen en yüksek doğruluk oranı %99,75. [11]

Kaggle üzerinden çeşitli yöntemler ile bu çözümlemeyi gerçekleştirmeye çalışmış insanların çalışmalarını inceledim. %100'e ulaşlığını iddia eden çeşitli kullanıcılar olsa da, bu tarz kesinlikte bir doğruluk oranının olası aşırı eğitilme durumuyla ilişkili olabileceğini düşünmekteyim. Nispeten basit CNN modelleri kullanılarak %98 civarında doğruluk düzeyine ulaşabilen örnekler mevcut. [12] Gözlemlerime göre, genellikle yeteri kadar etkin ve güçlü dizayn edilmiş bir sinir ağı söz konusuysa doğruluk oranları %99 ve üzerinde oluyor. Yetersiz eğitim süresi, yetersiz nöron sayısı, yetersiz katman sayısı, az sayıda filtre gibi çeşitli etkenler doğruluk oranını negatif yönde etkileyebiliyor.

İncelediğim kaynaklara göre temel farklarımın biri, projede kullandığım evrişimsel yapay sinir ağı modelini tasarlarken yüksek sayıda evrişim katmanı kullanmam. Bu evrişim katmanlarında da bilgisayarımın işlem gücünün el verdiği kadar yüksek sayıda filtre kullanmaya çalıştım. Bir sonraki bölümde de bahsedeceğim üzere, filtre sayısını düşük tutmak modelin doğruluk düzeyini negatif yönde etkiledi.

**Not:** Yukarıda belirttiğim kaynaklara ek olarak, keras, matplotlib ve sklearn kütüphanesinin dökümentasyonlarından da kaynak olarak faydalandım.

**4. Eğitim işlemini nasıl yaptığınızı (K-Fold CV / Eğitim / Test verisi sayısı, ... ) raporda anlatınız. Katman sayısı, epok sayısı, filtre boyutları, seyreltme oranı ve diğer parametre değerlerini değiştirerek sonuçlarınızı iyileştirmeye çalışınız. Hazır bir araç ile hiperparametre optimizasyonu da yapabilirsiniz. Parametre değerlerine karşılık aldığınız Loss, Accuracy gibi başarı sonuçlarını içeren bir tablo ve grafik oluşturunuz. Rapor ekleyiniz. Başarımı artırmak için hangi yöntemleri kullandığınızı ve ne kadar artış olduğunu belirtiniz. (25)**

#### **4.4.1 Eğitim İşleminin Anlatımı**

Eğitimde toplamda 2 farklı yöntem kullandım. Birincisi herhangi bir K-Fold çapraz doğrulama yöntemi kullanmadan eğitim gerçekleştirmek; diğeri de 5 katmanlı çapraz doğrulama kullanılarak eğitim gerçekleştirmekti. Esasen, 5 katmanlı çapraz doğrulamayı birincil olarak kullanmak isterdim ancak yüksek epok sayısı ve modelin karmaşıklığı gibi çeşitli etkenlerden dolayı bu durum eğitim süresini aşırı uzatma gibi bir soruna sebep oldu. Elbette daha güçlü bir sistem ile daha ideal bir eğitim süreci gerçekleştirilebilir. 5 katlı çapraz doğrulamayı kabul edilebilir sürelerde gerçekleştirebilmek için epok sayısını 12'ye indirmek zorunda kaldım. Oysa tek tur eğitim gerçekleştirdiğim diğer eğitim metodunda epok sayısı 30 turdu. Bu sayede toplam eğitim süresini normal ve kabul edilebilir bir düzeye indirmeyi başardım.

Veri setinde toplam 60.000 adet eğitim görseli ve 10.000 adet test görseli bulunmaktaydı. Öncelikli olarak bu görsellerin ölçeklenmesi ve normalize edilmesi için gerekli işlemleri gerçekleştirdim. Ardından kategorik etiketleme için gerekli işlemleri gerçekleştirdim. Eğitim sırasında validasyon verisi olarak test verilerini kullandım. 5 katmanlı çapraz doğrulama sırasında da ilk yöntemde olduğu gibi veri böülümlendirmesi yaptım ancak elbette çapraz doğrulamada her tur sırasında veriler çeşitli şekilde karıştırılıp farklı gruplar halinde seçildiler. Bu sayede modele yönelik doğruluk değerleri konusunda herhangi bir Bias olup olmadığını; dolayısıyla sonucun olması gerekenden iyi veya kötü olup olmadığını daha etkin bir şekilde sınamış oldum.

Verilerin eğitimi sırasında kullandığım model çok katmanlı bir evrişimsel sinir ağı modeliydi. Ayrıca modelin eğitimi sırasında gelişmeler oldukça sinir ağındaki ağırlıkların kaydedilebilmesini sağlayacak Callback metodlarını kullandım. Loss fonksiyonu olarak Categorical Crossentropy, Optimizasyon fonksiyonu olarak da Adam Optimizasyonunu tercih ettim.

#### **4.4.2 Tablo ve Grafik (Parametre Optimizasyonu). Başarımı artttırmak için hangi yöntemleri kullandığınızı ve ne kadar arttığını belirtiniz.**

Başarımı artttırmak için kullandığım çeşitli teknikler ve denedigim yöntemler aşağıdaki gibidir.

- Evrişimsel sinir ağındaki katman sayısını artttırmak
- Evrişimsel katmanlardaki filtre sayılarının yüksek tutulması
- Batch Normalizasyon katmanlarının kullanılması
- Ezberlemeyi engellemek için Dropout katmanları kullanılması
- Görüntü tanımlama amacı için daha etkin olduğu düşünülen ReLU aktivasyon metotunun tercih edilmesi
- Veri setinin eğitime sokulmadan önce ölçeklendirilmesi
- Düzleştirme katmanı kullanılması
- Son katmanda olasılıksal çıkarımda daha etkin olacak Softmax aktivasyon metodu kullanılması.
- Sondan bir önceki Dense Katmanda yeterli olacak sayıda nöron kullanılması
- Performans verimliliği ve doğruluk arasında olabildiğince optimize edilmiş noktada bir batch boyutu kullanmak.

Model, ilk versiyonunda 2 katman evrişimsel katmana ve ardından gelen bir havuz katmana sahipti. Bunun ardından düzleştirme katmanı ve 64 nörondan oluşan bir yoğun katman geliyordu. Bu halindeyken performansı daha düşüktü. (~%88)

Filtre sayıları da 16 olup modelin son haline göre daha düşüktü. Filtre sayılarını artttırmak ve 32'ye çıkarmak modelin performansında küçük miktarda iyileşmeye yol açtı. (~%89) Ardından yoğun katmandaki nöron sayısını artttırmayı denedim. 128, 256, 512 gibi çeşitli değerler, eğitim süresinde uzamaya yol açarken doğruluk değeri üzerindeki değişim anlamlı boyutta sadece 128'de meydana geldi.

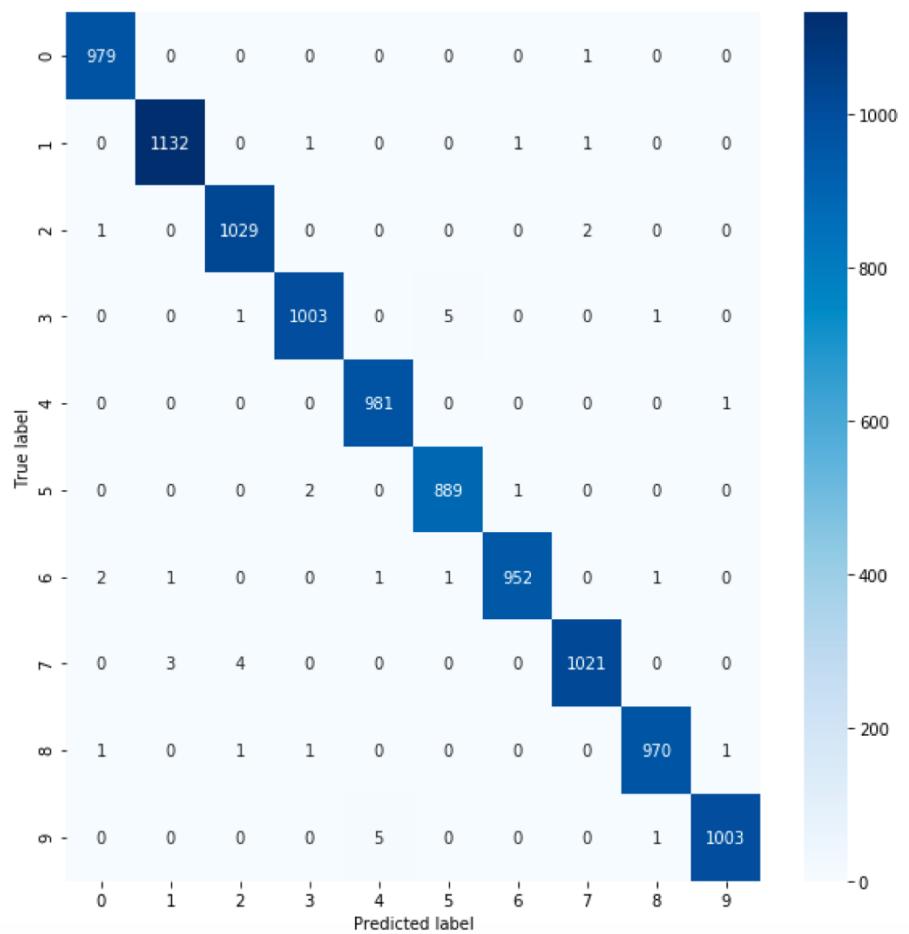
Son katman için 128 nöron sayısının uygun olduğu kanaatine vardım. Ardından evrişimsel katmanların sayısını artırmamanın verimli olabileceğü düşüncesine dayanarak uygulamaya koyuldum. Toplamda 6 adet evrişimsel katmanın bulunduğu yeni bir model denedim ve doğruluk oranında ciddi ilerlemeler gördüm. İlk 3 katman için 32, ardından 3 katman için 64 filtre kullandım. Doğruluk oranları ulaştığım en yüksek sonuç olan %99.62'ye yaklaşmaya başladım. Batch normalizasyon katmanları ve Dropout katmanları kullanarak modelin aşırı öğrenme, ezberleme gibi durumları minimize ederken performansını maksimize etmek konusunda çaba gösterdim.

Batch sayısını daha düşük tutmayı denediğimde eğitim süresinde ciddi uzamalara tanık oldum. Buna ek olarak çok fazla artırdığımda ise belli bir noktadan sonra yine eğitimin yavaşlamaya başladığını gördüm. Bu sebepten dolayı 32 değerinin kullanılmaya uygun olduğunu karar kıldım.

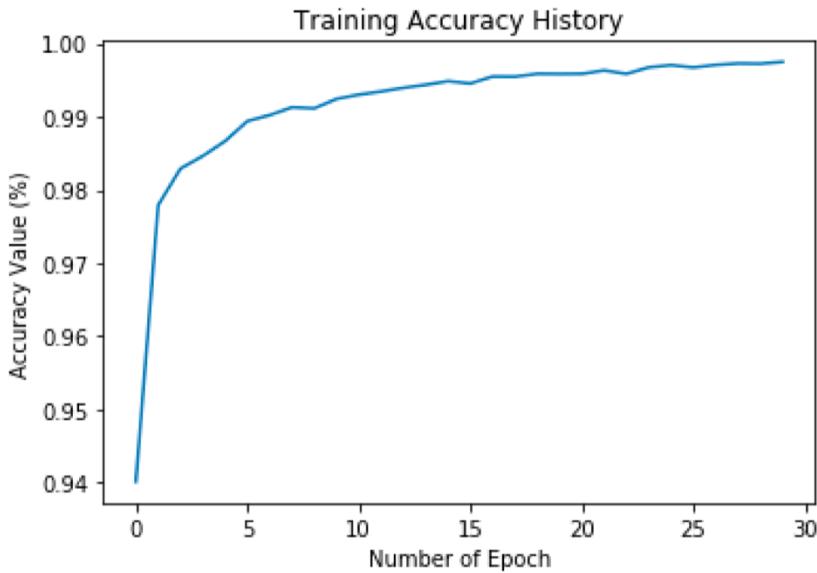
Eğitim süresi konusunda ise öncelikli olarak 10 epok boyunca eğitim gerçekleştirilmesini denedim. Fakat 10. Epok'un sonunda veri seti öğrenimini hala gerçekleştirdiğinden sonrasında 20 epok denedim. Ufak ilerlemeler olsa da hala öğrenme faaliyetinin gerçekleştiğini gördüm ve bu sebepten ötürü 30 epok kullanmak konusunda karar kıldım.

Bütün bu düzenlemeler ve ayarlamalar göz önüne alındığında modelin ilk versiyonu ile son versiyonu arasında ~%11 kadar bir doğruluk oranı artışı sağladığımı söyleyebilirim.

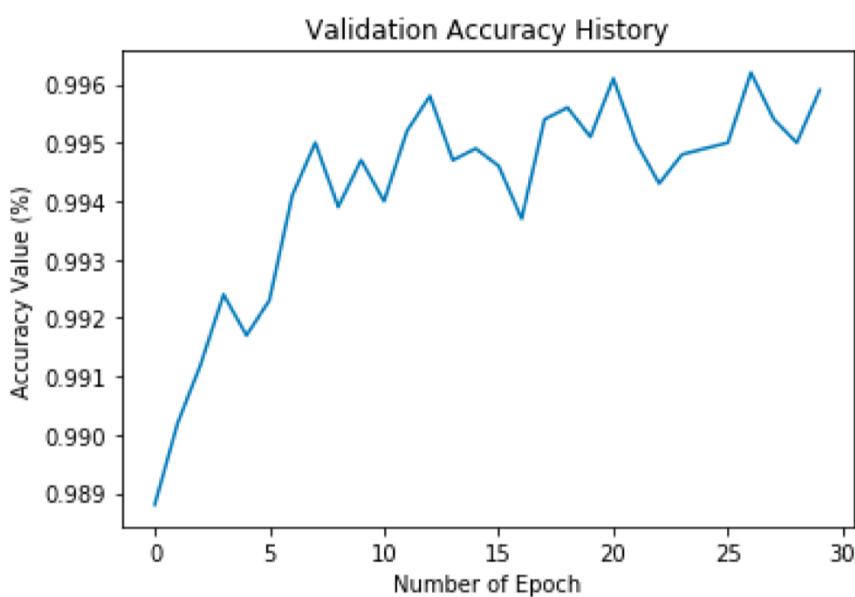
Aşağıda eğitim sürecine ait çeşitli grafiklerin ve görselleştirmelerin yer aldığı şekilleri inceleyebilirsiniz.



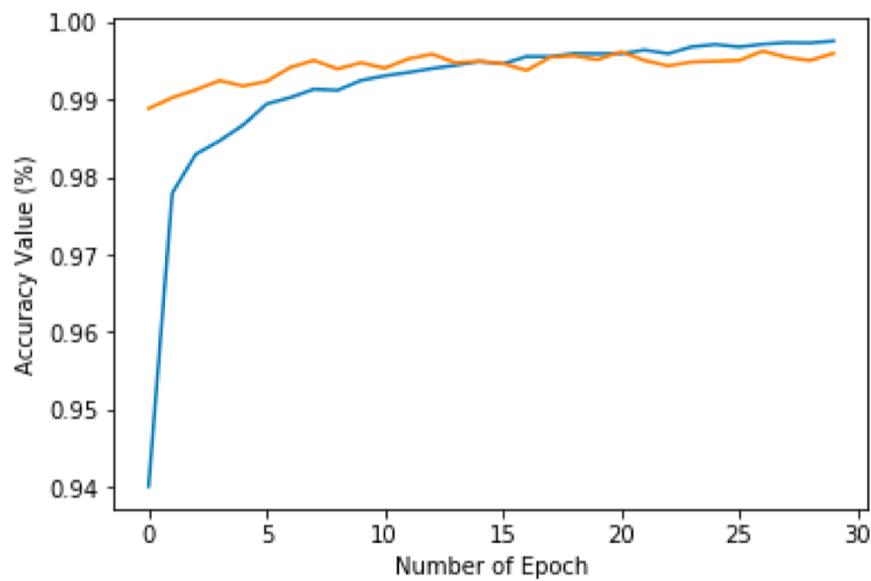
**Görsel 16:** Eğitim süreci sonrası test seti üzerinde yapılan değerlendirmen elde edilen hata matrisi



**Görsel 17:** Eğitim setine ait doğruluk değerinin eğitim turları süresince değişiminin grafiği



**Görsel 18:** Test setine ait doğruluk değerinin eğitim turları süresince değişiminin grafiği



**Görsel 19:** Eğitim ve Test setine ait doğruluk değerinin görselleştirilmesi

```

10000/10000 [=====] - 5s 514us/step
Model Loss : 0.017260911832495528
Model Accuracy : % 99.62000250816345

```

**Görsel 20:** Eğitim sonucunda ulaşılan Loss ve Accuracy değerlerinin gösterildiği görüntü

```

Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [=====] - 181s 3ms/step - loss: 0.1964 - accuracy: 0.9401 -
val_loss: 0.0344 - val_accuracy: 0.9888

Epoch 00001: val_accuracy improved from -inf to 0.98880, saving model to model.hdf5
Epoch 2/30
60000/60000 [=====] - 187s 3ms/step - loss: 0.0721 - accuracy: 0.9779 -
val_loss: 0.0328 - val_accuracy: 0.9902

Epoch 00002: val_accuracy improved from 0.98880 to 0.99020, saving model to model.hdf5
Epoch 3/30
60000/60000 [=====] - 187s 3ms/step - loss: 0.0560 - accuracy: 0.9829 -
val_loss: 0.0266 - val_accuracy: 0.9912

Epoch 00007: val_accuracy improved from 0.99240 to 0.99410, saving model to model.hdf5
Epoch 8/30
60000/60000 [=====] - 181s 3ms/step - loss: 0.0305 - accuracy: 0.9913 -
val_loss: 0.0176 - val_accuracy: 0.9950

Epoch 00008: val_accuracy improved from 0.99410 to 0.99500, saving model to model.hdf5
Epoch 9/30
60000/60000 [=====] - 178s 3ms/step - loss: 0.0296 - accuracy: 0.9912 -
val_loss: 0.0193 - val_accuracy: 0.9939

Epoch 00009: val_accuracy did not improve from 0.99500
Epoch 10/30
60000/60000 [=====] - 179s 3ms/step - loss: 0.0243 - accuracy: 0.9924 -
val_loss: 0.0181 - val_accuracy: 0.9947

```

**Görsel 21:** *Eğitim sürecinden bir görüntü*

```

Epoch 00014: val_accuracy did not improve from 0.99580
Epoch 15/30
60000/60000 [=====] - 181s 3ms/step - loss: 0.0172 - accuracy: 0.9949 -
val_loss: 0.0176 - val_accuracy: 0.9949

Epoch 00015: val_accuracy did not improve from 0.99580
Epoch 16/30
60000/60000 [=====] - 180s 3ms/step - loss: 0.0178 - accuracy: 0.9945 -
val_loss: 0.0177 - val_accuracy: 0.9946

Epoch 00016: val_accuracy did not improve from 0.99580
Epoch 17/30
60000/60000 [=====] - 182s 3ms/step - loss: 0.0149 - accuracy: 0.9955 -
val_loss: 0.0227 - val_accuracy: 0.9937

Epoch 00022: val_accuracy did not improve from 0.99610
Epoch 23/30
60000/60000 [=====] - 183s 3ms/step - loss: 0.0134 - accuracy: 0.9959 -
val_loss: 0.0211 - val_accuracy: 0.9943

Epoch 00023: val_accuracy did not improve from 0.99610
Epoch 24/30
60000/60000 [=====] - 182s 3ms/step - loss: 0.0101 - accuracy: 0.9967 -
val_loss: 0.0226 - val_accuracy: 0.9948

Epoch 00024: val_accuracy did not improve from 0.99610
Epoch 25/30
60000/60000 [=====] - 185s 3ms/step - loss: 0.0102 - accuracy: 0.9970 -
val_loss: 0.0214 - val_accuracy: 0.9949

```

**Görsel 22:** *Eğitim sürecinden bir görüntü*

Yukarıda belirtilenlere ek olarak, bir de daha düşük epok sayısının kullanıldığı, fakat 5 kat çapraz doğrulama gerçekleştirilen bir eğitim de uygulanmıştır. Bu eğitimde kullanılan model aynı olup, sadece epok sayısı üzerinde farkilaştırma uygulanmıştır. Farklılaştırma gerçekleştirilmesinin temel sebebi, önceki gibi 30 epok ile eğitim gerçekleştirmenin eğitim süresinde aşırı artışa sebep olmasıdır. Bu nedenle, 30 yerine 12 epok eğitim gerçekleştirdim.

Aşağıda çapraz doğrulama kullanarak eğitim gerçekleştirdiğim Python dökümanına (kfold.py) ait kod görsellerini paylaşıyorum.

```
19 #Import the necessary libraries
20 import keras
21 from keras.datasets import mnist
22 import matplotlib.pyplot as plt
23 from keras import backend as K
24 from model import get_model
25 from keras.models import load_model
26 import os
27 import tensorflow as tf
28 import seaborn as sns
29 import numpy as np
30 from sklearn.model_selection import KFold
31
32
33 #Determine the batch size, number of classes and total epochs
34 batch_size = 32
35 num_classes = 10
36 epochs = 12
37
38 #height and width of the images
39 rows, cols = 28, 28
40
```

Görsel 23: kfold.py dökümanına ait kodun bir görüntüüsü

```

42 #Get the dataset
43 (x_train, y_train), (x_test, y_test) = mnist.load_data()
44
45
46 #Format the data for the channels
47 if K.image_data_format() == 'channels_first':
48     x_train = x_train.reshape(x_train.shape[0], 1, rows, cols)
49     x_test = x_test.reshape(x_test.shape[0], 1, rows, cols)
50     input_shape = (1, rows, cols)
51
52 else:
53     x_train = x_train.reshape(x_train.shape[0], rows, cols, 1)
54     x_test = x_test.reshape(x_test.shape[0], rows, cols, 1)
55     input_shape = (rows, cols, 1)
56
57
58 #Preprocess and normalize the data
59 x_train = x_train.astype('float32')
60 x_test = x_test.astype('float32')
61
62 x_train = x_train / 255
63 x_test = x_test / 255
64

```

**Görsel 24:** *kfold.py* dökümanına ait kodun bir görüntüüsü

```

^.
65 y_train = keras.utils.to_categorical(y_train, num_classes)
66 y_test = keras.utils.to_categorical(y_test, num_classes)
67
68
69 #Define inputs and targets
70 inputs = np.concatenate((x_train, x_test), axis=0)
71 targets = np.concatenate((y_train, y_test), axis=0)
72
73
74 #Define K-Fold Cross validation (10 fold)
75 kfold = KFold(n_splits=5, shuffle=True)
76
77 lsss = []
78 accs = []
79

```

**Görsel 25:** *kfold.py* dökümanına ait kodun bir görüntüüsü

```

79
80 #Start training and record scores
81 fold_no = 1
82 for train, test in kfold.split(inputs, targets):
83
84     model = get_model(input_shape)
85
86     history = model.fit(inputs[train], targets[train],
87                         batch_size = batch_size,
88                         epochs = epochs,
89                         verbose = 1
90                         )
91
92     scores = model.evaluate(inputs[test], targets[test], verbose = 1)
93     print("_____")
94     print("Fold ", fold_no, " Loss is : ", scores[0])
95     print("Fold ", fold_no, " Accuracy Score is : ", scores[1]*100, "%")
96     print("_____")
97
98     lsss.append(scores[0])
99     accs.append(scores[1])
100
101    fold_no = fold_no + 1
102

```

**Görsel 26:** *kfold.py* dökümanına ait kodun bir görsüntüsü

```

103
104 #Print average loss and average accuracy for the k-fold validation
105 print("_____")
106 print("AVERAGE LOSS : ", (sum(lsss)/len(lsss)))
107 print("AVERAGE ACCURACY : ", (sum(accs)/len(accs)))
108 print("_____")
109
110
111

```

**Görsel 27:** *kfold.py* dökümanına ait kodun bir görsüntüsü

5 katlı çapraz doğrulama uygulandıktan sonra elde edilen doğruluk oranlarına yönelik çıktılar aşağıdaki gibidir.

1. ~%99.40
2. ~%99.33
3. ~%99.38
4. ~%99.47
5. ~%99.34

**Ortalama:** ~%99.38 ± 0.09

Doğruluk değerleri incelendiğinde, modelin yüksek oranda Bias taşıdığına yönelik herhangi bir kuşkuya sebep olmadığı görülmektedir. Sonuçlar birbirine yakındır.

K-Fold çapraz doğrulama olmadan gerçekleştirilen önceki eğitimin sonucu olan %99,62 ile arasında yer alan ~%0,24'lük farkın sebebi muhtemelen çapraz doğrulamayı mümkün kılma gayesiyle eksiltilen 18 epok'tur. (Donanım işlem gücü kısıtlayıcı faktör.)

Aşağıda 5 kat çapraz doğrulama kullanılarak gerçekleştirilen eğitimin konsol çıktılarını inceleyebilirsiniz.

```
user@user-Sun:/Desktop/PycharmProjects/ML/ML_competitions/kaggle/Reloaded modules: model
Epoch 1/12
56000/56000 [=====] - 139s 2ms/step - loss: 0.2122 - accuracy: 0.9367
Epoch 2/12
56000/56000 [=====] - 141s 3ms/step - loss: 0.0719 - accuracy: 0.9784
Epoch 3/12
56000/56000 [=====] - 143s 3ms/step - loss: 0.0586 - accuracy: 0.9823
Epoch 4/12
56000/56000 [=====] - 141s 3ms/step - loss: 0.0501 - accuracy: 0.9847
Epoch 5/12
56000/56000 [=====] - 142s 3ms/step - loss: 0.0436 - accuracy: 0.9873
Epoch 6/12
56000/56000 [=====] - 143s 3ms/step - loss: 0.0375 - accuracy: 0.9884
Epoch 7/12
56000/56000 [=====] - 142s 3ms/step - loss: 0.0326 - accuracy: 0.9896
Epoch 8/12
56000/56000 [=====] - 144s 3ms/step - loss: 0.0303 - accuracy: 0.9907
Epoch 9/12
56000/56000 [=====] - 154s 3ms/step - loss: 0.0283 - accuracy: 0.9914
Epoch 10/12
56000/56000 [=====] - 148s 3ms/step - loss: 0.0245 - accuracy: 0.9923
Epoch 11/12
56000/56000 [=====] - 148s 3ms/step - loss: 0.0233 - accuracy: 0.9932
Epoch 12/12
56000/56000 [=====] - 148s 3ms/step - loss: 0.0222 - accuracy: 0.9932
14000/14000 [=====] - 6s 428us/step

Fold 1 Loss is : 0.021095743231559546
Fold 1 Accuracy Score is : 99.40000176429749 %
```

**Görsel 28:** *kfold.py* eğitim sürecine ait bir görüntü

```

Epoch 1/12
56000/56000 [=====] - 149s 3ms/step - loss: 0.1938 - accuracy: 0.9408
Epoch 2/12
56000/56000 [=====] - 148s 3ms/step - loss: 0.0695 - accuracy: 0.9791
Epoch 3/12
56000/56000 [=====] - 151s 3ms/step - loss: 0.0561 - accuracy: 0.9830
Epoch 4/12
56000/56000 [=====] - 149s 3ms/step - loss: 0.0508 - accuracy: 0.9848
Epoch 5/12
56000/56000 [=====] - 150s 3ms/step - loss: 0.0418 - accuracy: 0.9873
Epoch 6/12
56000/56000 [=====] - 151s 3ms/step - loss: 0.0366 - accuracy: 0.9889
Epoch 7/12
56000/56000 [=====] - 151s 3ms/step - loss: 0.0331 - accuracy: 0.9899
Epoch 8/12
56000/56000 [=====] - 153s 3ms/step - loss: 0.0299 - accuracy: 0.9911
Epoch 9/12
56000/56000 [=====] - 149s 3ms/step - loss: 0.0272 - accuracy: 0.9921
Epoch 10/12
56000/56000 [=====] - 150s 3ms/step - loss: 0.0249 - accuracy: 0.9927
Epoch 11/12
56000/56000 [=====] - 150s 3ms/step - loss: 0.0237 - accuracy: 0.9928
Epoch 12/12
56000/56000 [=====] - 150s 3ms/step - loss: 0.0203 - accuracy: 0.9939
14000/14000 [=====] - 6s 438us/step

```

```

Fold 2 Loss is : 0.02177007615146795
Fold 2 Accuracy Score is : 99.32857155799866 %

```

**Görsel 29:** *kfold.py* *eğitim sürecine ait bir görüntü*

```

Epoch 1/12
56000/56000 [=====] - 153s 3ms/step - loss: 0.1898 - accuracy: 0.9419
Epoch 2/12
56000/56000 [=====] - 149s 3ms/step - loss: 0.0691 - accuracy: 0.9793
Epoch 3/12
56000/56000 [=====] - 150s 3ms/step - loss: 0.0565 - accuracy: 0.9823
Epoch 4/12
56000/56000 [=====] - 152s 3ms/step - loss: 0.0502 - accuracy: 0.9850
Epoch 5/12
56000/56000 [=====] - 150s 3ms/step - loss: 0.0422 - accuracy: 0.9872
Epoch 6/12
56000/56000 [=====] - 151s 3ms/step - loss: 0.0384 - accuracy: 0.9887
Epoch 7/12
56000/56000 [=====] - 152s 3ms/step - loss: 0.0341 - accuracy: 0.9898
Epoch 8/12
56000/56000 [=====] - 150s 3ms/step - loss: 0.0310 - accuracy: 0.9904
Epoch 9/12
56000/56000 [=====] - 150s 3ms/step - loss: 0.0287 - accuracy: 0.9912
Epoch 10/12
56000/56000 [=====] - 150s 3ms/step - loss: 0.0253 - accuracy: 0.9921
Epoch 11/12
56000/56000 [=====] - 151s 3ms/step - loss: 0.0236 - accuracy: 0.9929
Epoch 12/12
56000/56000 [=====] - 149s 3ms/step - loss: 0.0228 - accuracy: 0.9929
14000/14000 [=====] - 6s 430us/step

```

```

Fold 3 Loss is : 0.022130748603186087
Fold 3 Accuracy Score is : 99.38571453094482 %

```

**Görsel 30:** *kfold.py* *eğitim sürecine ait bir görüntü*

```

Epoch 1/12
56000/56000 [=====] - 152s 3ms/step - loss: 0.1900 - accuracy: 0.9421
Epoch 2/12
56000/56000 [=====] - 153s 3ms/step - loss: 0.0707 - accuracy: 0.9787
Epoch 3/12
56000/56000 [=====] - 152s 3ms/step - loss: 0.0583 - accuracy: 0.9823
Epoch 4/12
56000/56000 [=====] - 152s 3ms/step - loss: 0.0483 - accuracy: 0.9855
Epoch 5/12
56000/56000 [=====] - 151s 3ms/step - loss: 0.0420 - accuracy: 0.9876
Epoch 6/12
56000/56000 [=====] - 152s 3ms/step - loss: 0.0382 - accuracy: 0.9884
Epoch 7/12
56000/56000 [=====] - 152s 3ms/step - loss: 0.0345 - accuracy: 0.9896
Epoch 8/12
56000/56000 [=====] - 152s 3ms/step - loss: 0.0283 - accuracy: 0.9913
Epoch 9/12
56000/56000 [=====] - 150s 3ms/step - loss: 0.0298 - accuracy: 0.9909
Epoch 10/12
56000/56000 [=====] - 147s 3ms/step - loss: 0.0248 - accuracy: 0.9923
Epoch 11/12
56000/56000 [=====] - 147s 3ms/step - loss: 0.0229 - accuracy: 0.9926
Epoch 12/12
56000/56000 [=====] - 139s 2ms/step - loss: 0.0221 - accuracy: 0.9934
14000/14000 [=====] - 6s 411us/step

Fold 4 Loss is : 0.019535516330353858
Fold 4 Accuracy Score is : 99.47142601013184 %

```

**Görsel 31:** *kfold.py* eğitim sürecine ait bir görüntüyü

```

Epoch 1/12
56000/56000 [=====] - 145s 3ms/step - loss: 0.2037 - accuracy: 0.9376
Epoch 2/12
56000/56000 [=====] - 144s 3ms/step - loss: 0.0728 - accuracy: 0.9776
Epoch 3/12
56000/56000 [=====] - 152s 3ms/step - loss: 0.0583 - accuracy: 0.9824
Epoch 4/12
56000/56000 [=====] - 150s 3ms/step - loss: 0.0499 - accuracy: 0.9858
Epoch 5/12
56000/56000 [=====] - 149s 3ms/step - loss: 0.0423 - accuracy: 0.9868
Epoch 6/12
56000/56000 [=====] - 151s 3ms/step - loss: 0.0368 - accuracy: 0.9886
Epoch 7/12
56000/56000 [=====] - 150s 3ms/step - loss: 0.0324 - accuracy: 0.9899
Epoch 8/12
56000/56000 [=====] - 151s 3ms/step - loss: 0.0305 - accuracy: 0.9907
Epoch 9/12
56000/56000 [=====] - 151s 3ms/step - loss: 0.0269 - accuracy: 0.9920
Epoch 10/12
56000/56000 [=====] - 151s 3ms/step - loss: 0.0250 - accuracy: 0.9921
Epoch 11/12
56000/56000 [=====] - 152s 3ms/step - loss: 0.0221 - accuracy: 0.9929
Epoch 12/12
56000/56000 [=====] - 152s 3ms/step - loss: 0.0209 - accuracy: 0.9938
14000/14000 [=====] - 6s 443us/step

Fold 5 Loss is : 0.02417603481921417
Fold 5 Accuracy Score is : 99.34285879135132 %

```

**Görsel 32:** *kfold.py* eğitim sürecine ait bir görüntüyü

---

**AVERAGE LOSS :** 0.021741623827156324  
**AVERAGE ACCURACY :** 0.9938571453094482

---

**Görsel 33:** *kfold.py* eğitim sürecine ait bir görüntüyü

#### **4.4.3 Yorumlar (En başarılı parametre değerleri, nedenleri)**

Batch sayısı açısından en başarılı parametre değerini 32 olarak belirledim. Bunun en temel sebeplerinden biri batch sayısını çok yüksek tutmanın eğitim hızı açısından negatif geri dönüşe sebep olması, çok düşük tutmanın da yine aynı şekilde negatif etkiye yol açmasıydı. 32 ise optimum hızda işlenmesini sağlıyordu. Filtre sayısının daha da arttırılmasının performansı olumlu etkileyebileceğini düşünmekteyim ancak kulandığım donanım buna elverecek kadar güçlü olmadığından 3 adet 32 filtreli, 3 adet de 64 filtreli evrişimsel katman kulandım.

Buna ek olarak, Dropout değeri üzerinde çeşitli oynamalarda da bulundum. 0.2, 0.3 ve 0.4 dropout değerleri denedim ve aralarında kayda değer doğruluk farklarına rastlamadım. Fakat daha düşük seçmenin negatif etkilere sahip olabileceği düşüncesindeyim.

Daha fazla evrişimsel katman kullanmak avantaj sağlayabilir fakat yine donanımla ilgili sebeplerden ötürü katman sayısını fazla sayıda tutamadım. Yoğun katmandaki nöron sayısını daha fazla tutmak, örneğin 256 veya 512 olarak belirlemek doğrulukta herhangi bir pozitif değişim sebep olmadı. Ancak 32 ve 64 gibi daha düşük değerler negatif etkiye sebep oldu. Bu yüzden 128'i uygun bir değer olarak belirledim.

Görüntü işlemeyle ilgili modellerde performansa olumlu etki yaptığı düşünülen ReLU aktivasyon fonksiyonunu kullanmanın avantaj sağlayabileceğini düşündüm. Son olarak epok sayısını önce 10 olarak belirlemiştim. Fakat 10 turun sonunda model hala öğrenmesine devam edebiliyordu bu sebeple önce 20'ye, sonra yine aynı sebeple 30'a çıkardım. Daha kuvvetli bir donanımdan faydalananarak belki 35 veya 40 epok sonucunda doğrulukta değişim olup olmayacağıni görmek de faydalı olabilir.

## 5. Özdeğerlendirme Raporu (10)

	İstenen Özellik	Puan	Var	Açıklama	Tahmin i Puan
<b>0</b>	Kapak Sayfası	5	<input checked="" type="checkbox"/>	İstenen özelliklerde bir kapak sayfası geliştirildi.	<b>5</b>
<b>1</b>	Image Classification, Object Detection ve Image Segmentation Karşılaştırması	5	<input checked="" type="checkbox"/>	Bahsedilen kavramlar hakkında bilgi verildi ve aralarındaki farklar açıklandı.	<b>5</b>
<b>2</b>	Araştırma (Ön Çalışma)	5	<input checked="" type="checkbox"/>	Belirtilen konular hakkında araştırmalar yapıldı ve videolar izlendi. Öğrenilenler ve ilginç gelen detaylar raporda belirtildi.	<b>5</b>
<b>3.1</b>	Derin Öğrenme / Problemin Tanımı	5	<input checked="" type="checkbox"/>	Projede çözüm üretilen derin öğrenme problemine yönelik bilgi verildi ve amaçtan bahsedildi.	<b>5</b>
<b>3.2</b>	Derin Öğrenme / Veri Seti Hakkında Bilgi	5	<input checked="" type="checkbox"/>	Projede kullanılan MNIST veri setine yönelik bilgi verildi.	<b>5</b>
<b>3.3.1</b>	Model Oluşturma / Kullanılan Ortam, Dil, Kütüphaneler ve Sürümüleri	5	<input checked="" type="checkbox"/>	Projede kullanılan programlama dili, programlama ortamı, kullanılan kütüphaneler ve bunların sürümleri hakkında bilgi verildi ve rapora eklendi.	<b>5</b>
<b>3.3.2</b>	Model Oluşturma / Derin Öğrenme ile bir tanıma / tespit uygulaması kodu. Modelin adı, katman sayısı, şeması / şekli	10	<input checked="" type="checkbox"/>	Proje dahilinde geliştirilen modele yönelik bilgiler sağlandı. Katman sayısı belirtildi ve katmanların özellikleri hakkında bilgi verildi. Şema, çizge şeklinde ifade edildi. Modeli ifade eden kodların görselleri paylaşıldı. Veri setinin eğitimini ve testini içeren kodlara ait görseller paylaşıldı.	<b>10</b>
<b>3.3.3</b>	Model Oluşturma / Hangi çalışma / hazır kod / bağlantidan faydalandınız? Faydalığınız kaynaklara göre farklılıklarınız nelerdir? Açıklayınız.	10	<input checked="" type="checkbox"/>	Benzer çalışmalarla olan farklılıklardan bahsedildi. Yararlanılan kaynaklara yönelik bilgi verildi. Kaynak numarası ile işaretlendi ve kaynakçada bu kaynaklara yönelik linkler belirtildi. Başarı oranları hakkında kısa bir tartışmada bulunuldu.	<b>10</b>
<b>4.4.1</b>	Eğitim işleminin anlatımı	5	<input checked="" type="checkbox"/>	Eğitim işleminin gerçekleştirildiği iki yöntem hakkında bilgi verildi. Bu yöntemlerin nasıl gerçekleştirildiği ve hangi teknolojilerin kullanıldığı anlatıldı.	<b>5</b>
<b>4.4.2</b>	Tablo ve Grafik (parametre optimizasyonu). Başarımı artırmak için hangi yöntemleri kullandığınızı ve artış miktarını belirtiniz.	15	<input checked="" type="checkbox"/>	Projede gerçekleştirilen eğitim sürecinin sonunda ulaşılan Loss ve Accuracy değerlerine yönelik grafikler rapora eklendi. Parametrelerin optimize edilmesi için gerçekleştirilen adımlardan ve yapılanlardan bahsedildi. Parametre optimizasyonu	<b>15</b>

				sonucunda başarıım oranlarındaki değişimler tartışıldı.	
<b>4.4.3</b>	Yorumlar (En başarılı parametre değerleri, nedenleri vs.)	5	<input checked="" type="checkbox"/>	En son kullanılan parametre değerlerinde neden karar kılındığı konusunda açıklamalar gerçekleştirildi. Başarıım üzerindeki etkileri hakkında tartışıldı ve bunların nedenleri konusunda fikir yürütüldü.	<b>5</b>
<b>5</b>	Özdeğerlendirme Raporu	10	<input checked="" type="checkbox"/>	Özdeğerlendirme raporu gereken şekilde dolduruldu.	<b>10</b>
<b>Ek.1</b>	Kaynakça	5	<input checked="" type="checkbox"/>	Kaynakça düzenlendi ve özellikle 3.3.3. maddede belirtilen ve faydalanan kaynaklar hakkında burada gerekli referanslar verildi.	<b>5</b>
<b>Ek.2</b>	Rapor Düzeni	10	<input checked="" type="checkbox"/>	Rapor açık ve anlaşılır bir düzende olacak şekilde tasarlantı ve kontrol edildi.	<b>10</b>
<b>Toplam</b>					<b>100</b>

**Not:** Raporda bahsedilen projeye ait kod dökümanları, raporun ilişğinde olduğu klasöre eklenmiştir.

**Not 2:** Kaynakça bir sonraki sayfada belirtilmiştir.

## **6. Kaynakça**

**[1]** What is Image Clasification?

<<https://desktop.arcgis.com/en/arcmap/latest/extensions/spatial-analyst/image-classification/what-is-image-classification-.htm>>

Accessed on 17 01 2021

**[2]** Object Detection.

<[https://en.wikipedia.org/wiki/Object\\_detection](https://en.wikipedia.org/wiki/Object_detection)> Accessed on 17 01 2021

**[3]** Image Segmentation.

<[https://en.wikipedia.org/wiki/Image\\_segmentation](https://en.wikipedia.org/wiki/Image_segmentation)> Accessed on 17 01 2021

**[4]** Keras API Documentation. <<https://keras.io/api/>> Accessed on 17 01 2021

**[5]** Keras CNN Example. <<https://www.kaggle.com/tonypoe/keras-cnn-example>> Accessed on 17 01 2021

**[6]** Building a Convolutional Neural Network (CNN) in Keras.

<<https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbad5f5>> Accessed 17 01 2021

**[7]** Matplotlib Documentation.

<<https://matplotlib.org/3.3.3/contents.html>> Accessed 17 01 2021

**[8]** Intuitively Understanding Convolutions for Deep Learning.

<<https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1>> Accessed 17 01 2021

**[9]** Max Pooling. <[https://computersciencewiki.org/index.php/Max-pooling\\_-\\_Pooling](https://computersciencewiki.org/index.php/Max-pooling_-_Pooling)> Accessed 17 01 2021

**[10]** Convolutional Neural Networks: An Intro Tutorial.  
<<https://heartbeat.fritz.ai/a-beginners-guide-to-convolutional-neural-networks-cnn-cf26c5ee17ed>> Accessed 17 01 2021

**[11]** How to Choose CNN Architecture MNIST.  
<<https://www.kaggle.com/cdeotte/how-to-choose-cnn-architecture-mnist>> Accessed 17 01 2021

**[12]** Digit Recognition – Notebooks.  
<<https://www.kaggle.com/c/digit-recognizer/notebooks>> Accessed 18 01 2021