T.C.

MUĞLA SITKI KOÇMAN ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY ZEKA ANABİLİM DALI

ZEKİ OPTİMİZASTON YÖNTEMLERİ FİNAL RAPORU

KONU: GENETİK ALGORİTMA İLE HİPER-PARAMETRELERİN AYARLANMASI

ÖĞRENCİ ADI-SOYADI: ERKAN EROL

ÖĞRENCİ NUMARASI: 214225002

İÇİNDEKİLER

1. GİRİŞ

Günümüzde çok sayıda gizli katmanlı derin ağların kullanılmaya başlamasıyla makine öğrenmesindeki çalışmalar özniteliklerin çıkarılmasından (bu kısım artık derin öğrenme modelleri tarafından gerçekleştirilmektedir) mimari yapının oluşturulmasına doğru kaymış durumdadır. Derin öğrenme yaklaşımlarının gelişimi ile çok katmanlı yapay sinir ağının nasıl tasarlanacağı; kaç katmandan oluşacağı, ne kadar nöron içereceği, hangi optimizasyon algoritmasının ya da aktivasyon fonksiyonunun kullanılacağı problem çözümünde daha önemli hale geldi.

Grafik kartları ile paralel işlemlerin daha hızlı yapılabilmesi araştırmacıların daha fazla katmandan oluşan daha karmaşık yapılar tasarlayabilmesine imkân vermesiyle birlikte; derin öğrenme ile problem çözmek çok katmanlı ağ yapısını en iyi ve optimum şekilde tasarlamak ile eşdeğer duruma geldi.

Bu çalışmada oluşturulan algoritma ile MNIST ve CIFAR-10 veri setleri üzerinden en yüksek doğrulukla sınıflandırma yapabilecek derin öğrenme modelinin genetik algoritma ile katman sayısı, katmanlardaki nöron sayısı, nöronlardaki aktivasyon fonksiyonu, kullanılacak optimizasyon fonksiyonu ve kullanılacak kayıp (loss) fonksiyonun belirlenmesi amaçlanmaktadır.

2. MATERYAL-METOD

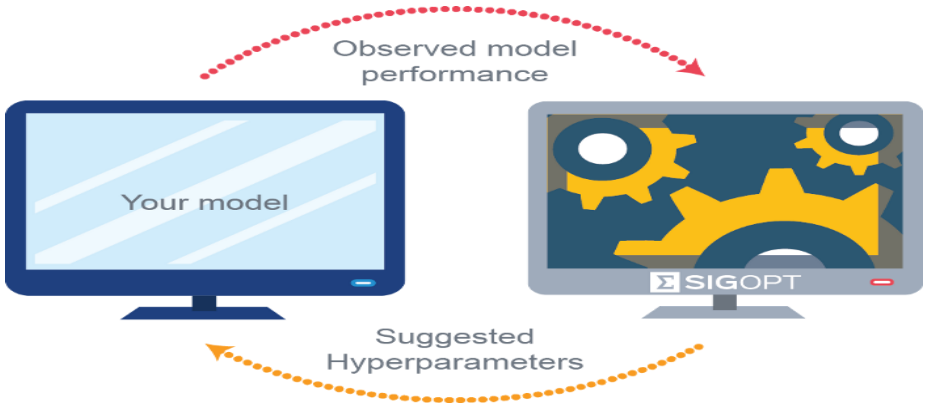
2.1. HİPER-PARAMETRE YÖNTEMLERİ

Makine öğrenmesi modelleri tasarlanırken modeli tasarlayan kişilere bırakılmış probleme, veri setine göre değişiklik gösteren parametreler hiper-parametre (hyperparameter) olarak adlandırılmaktadır.

Modelin farklı başarım sağladığı farklı hiper parametre grupları olabilmektedir. En uygun hiper parametre grubunun seçilmesi baş edilmesi gereken önemli problemlerden biridir. Son zamanlarda problemin çözümü için en uygun hiper parametre grubunun en optimum şekilde seçilmesine yönelik farklı teknikler ortaya atılmaktadır.

Sezgisel Parametre Ayarlama

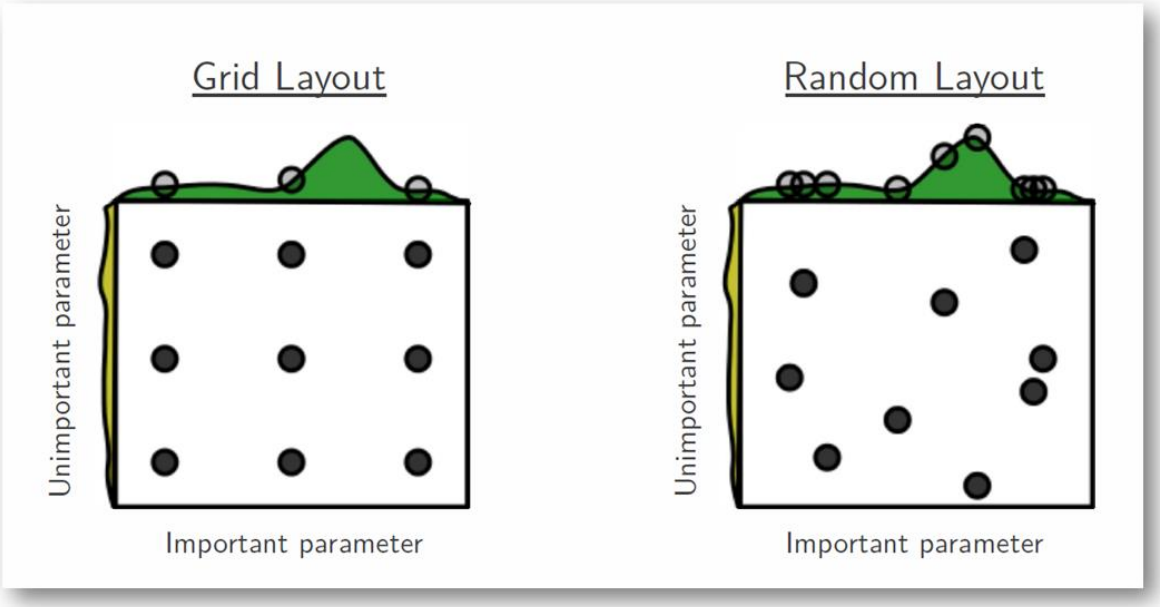
Probleme dair ön bilgilerimizi kullanarak hiper parametreler tahmin edilir, model bu hiper parametrelere göre tasarlanır ve sonuçlar gözlenir. Çıkan sonuçlara göre sezgisel olarak modelin başarımını artıracak yeni hiper parametre tahminleri yapılarak model tekrar oluşturulup, eğitilir ve sonuçlar gözlenir. Bu döngü Şekil 1'de de görüleceği üzere beklenen başarımı verecek uygun parametre grupları bulunana kadar devam eder.



Şekil-1 Sezgisel Arama

Grid Search ile Uygun Parametreleri Bulma

Hiper parametrelerden bazıları sonsuz sayıda değer alabilecek konumdadırlar. Bununla birlikte biz problem hakkında sahip olduğumuz ön bilgileri kullanarak hiper parametrelerin alabilecekleri değerler için aralıklar belirleyebiliriz. Belirlediğimiz bu aralıklardan belirli ana noktalar seçilerek hiper parametreler için değer listeleri oluşturulur. Grid search ile hiper parametre seçim işleminde; belirlenen aralıkta bulunan tüm değerlerin kombinasyonları için ağ eğitilip sonuçlar gözlenir duruma göre en iyi kombinasyon hiper parametre grubu olarak seçilir.

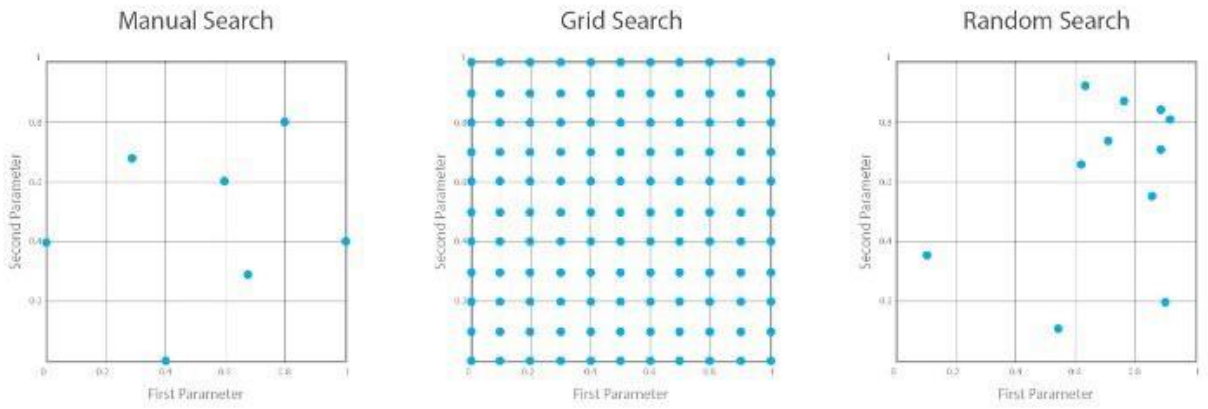


Şekil-2 Grid Search

Random Search ile Parametre Arama

İlk olarak Bengio tarafından 2012 yılında önerilmiştir. Grid search'te olduğu gibi probleme dair ön bilgiler kullanılarak hiper parametre aralıkları belirlenir. Daha sonra bu aralıktaki değerlerin her birini denemek yerine rastgele değerler seçilerek hiper parametre grupları

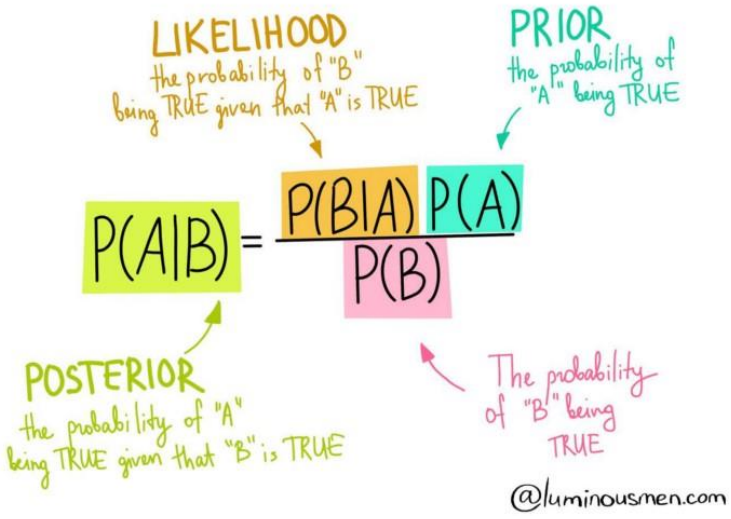
oluşturulur. Daha sonra en iyi sonucu bulana kadar veya sonuca dair beklenen değere ulaşana kadar rastgele farklı parametre gruplarıyla model eğitilerek başarım gözlenir. Sonuçlara göre en uygun hiper parametre grubu bulunur.



Şekil-3 Random Search

Bayes Yöntemi ile Hiper Parametre Arama

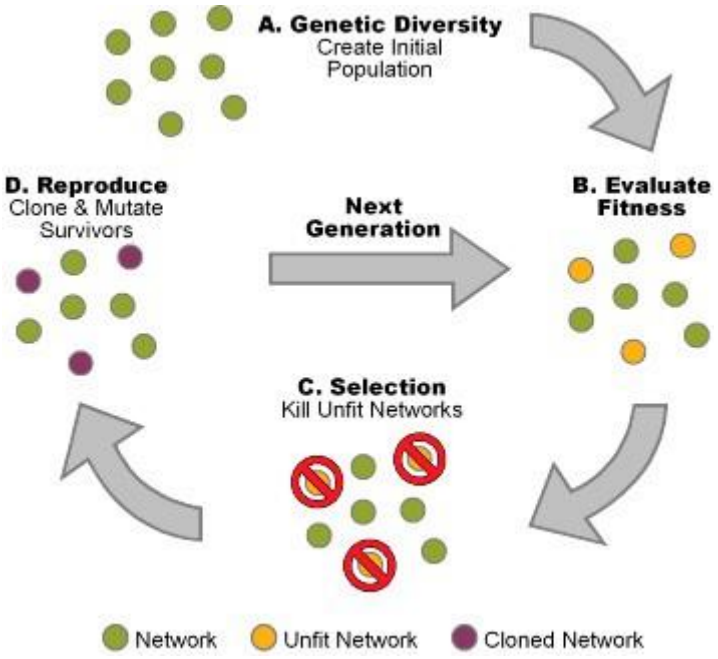
Ryan Adams tarafından 2012 yılında önerilen ve Google Cloud tarafından da kullanılan yöntemde hiper parametreler oluşturularak yapılan çalışmaların sonuçları bir sonraki hiper parametre seçiminde kullanılmaktadır. Bu seçim işleminde bilinen Bayes teoremi üzerinden olasılık hesabı yapılmaktadır.



Şekil-4 Bayes Teoremi

Genetik Algoritma ile Parametre Ayarlama

Rastgele aramaya benzer şekilde çalışır. Belli parametreler sabit tutularak her denemede çok az kısım parametrenin değeri değiştirilerek model eğitilip sonuçlar gözlenir.

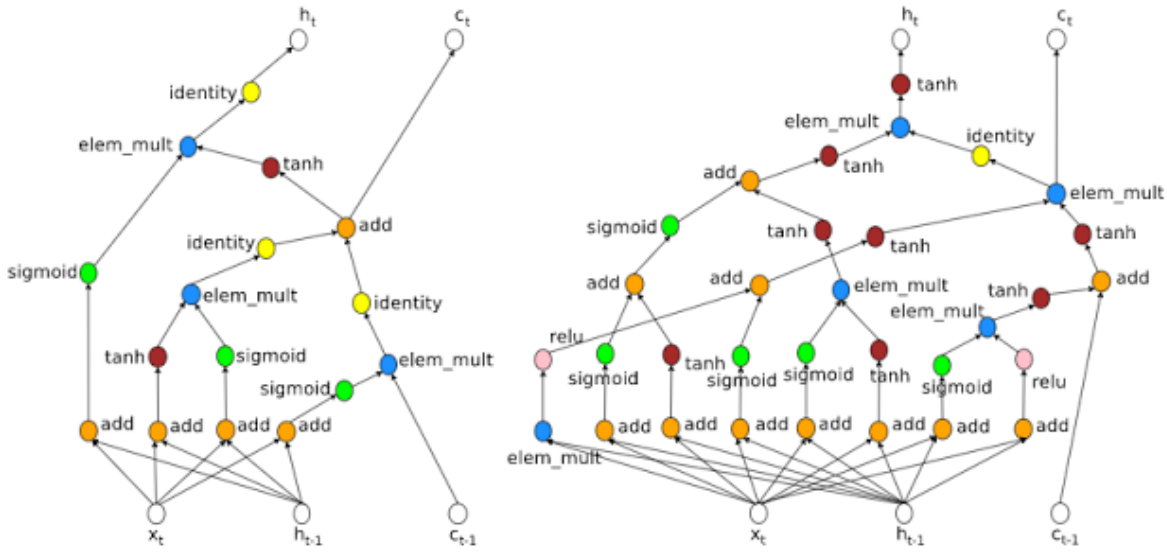


Şekil-5 Genetik Algoritma Seçim Döngüsü

Derin Öğrenme ile Hiper Parametre Öğrenme

Son dönemde daha önceki derin öğrenme modellerini öğrenerek oluşturulacak yeni modellerin parametrelerini hesaplayan modeller üzerine çalışılmaktadır. Google Brain ekibi tarafından oluşturulmaya çalışılmaktadır. Derin öğrenme ile model oluşturacak model tasarlanması projesinde; derin öğrenme modeli tarafından hiper parametreler seçilerek modeller tasarlanmaktadır. Yapılan çalışmalarda bu yöntem ile daha hızlı ve daha kompleks modeller oluşturulduğu gözlenmiştir. Bununla birlikte oluşturulan modellerde fazladan katmanların ve bağların olduğu da gözlenmiş. Bu bağ ve katmanların hata mı yoksa iyileştirme mi olduğu henüz

tanımlanamamıştır. Yapılan çalışmaların sonuçları hala tartışamaya açık durumdadır. Bu teknik hala gelişme aşamasındadır.



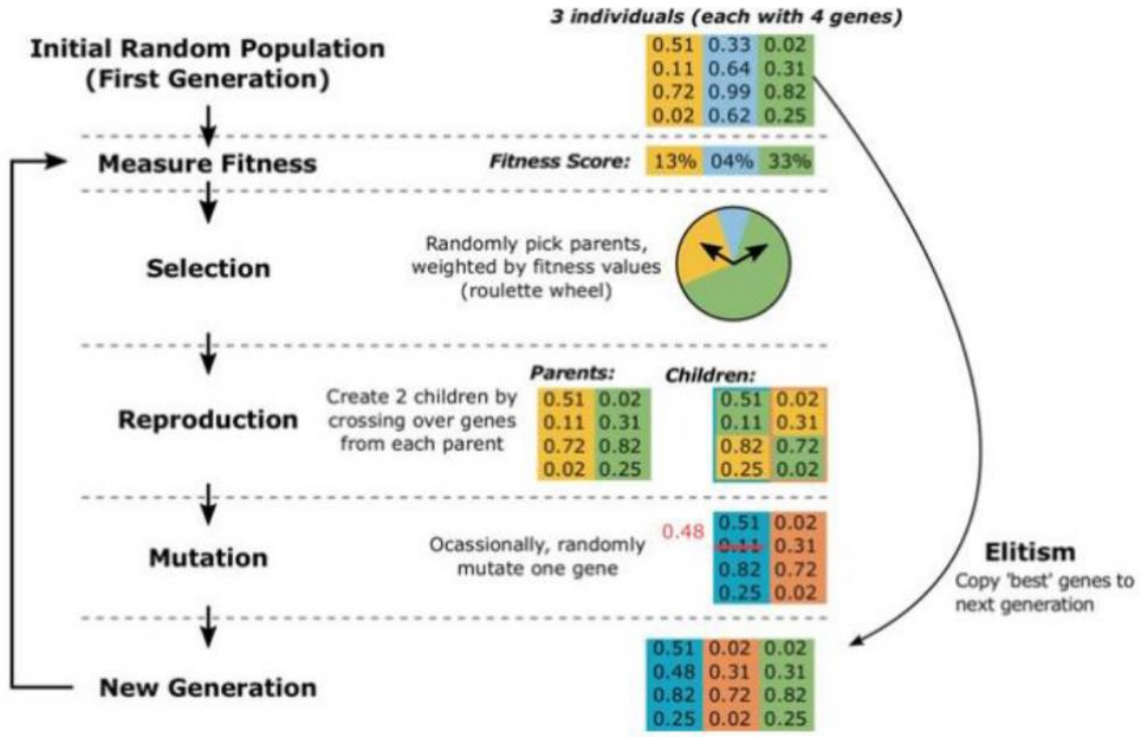
Şekil-6 Google tarafından oluşturulan CIFAR modeli ve derin öğrenme modeli tarafından üretilen CIFAR modeli

2.2 GENETİK ALGORİTMA

Yapay Zeka alanında kullanılan Genetik Algoritma, bir tür en iyi noktayı arayan algoritmadır. Bir probleme çözüm aramakla ilgilenir. Gerçek hayatta karşılaştığımız bir problemi (kuru yük gemisine konteynerler nasıl yerleştirilmeli, bir noktadan başka bir noktaya nasıl gidilebilir ya da en uygun teslimat rotası oluşturma gibi), biz bir arama problemine dönüştürebilirsek, bu problemi Genetik Algoritma ile çözebiliriz.

Genetik Algoritma (GA), permütasyon tabanlı bir optimizasyon yapar ve olasılıklar üzerinden yakınsama kriterleri altında arama yapan bir fonksiyondur. Doğada gözlemlenen evrimsel sürece benzer bir şekilde çalışan, arama ve eniyileme yöntemidir. Genetik Algoritma literatürde şöyle açıklanmıştır: Genetik Algoritma, biyolojik evrimin temel prensiplerinden ilham alan güçlü bir evrimsel stratejidir. Araştırmacı

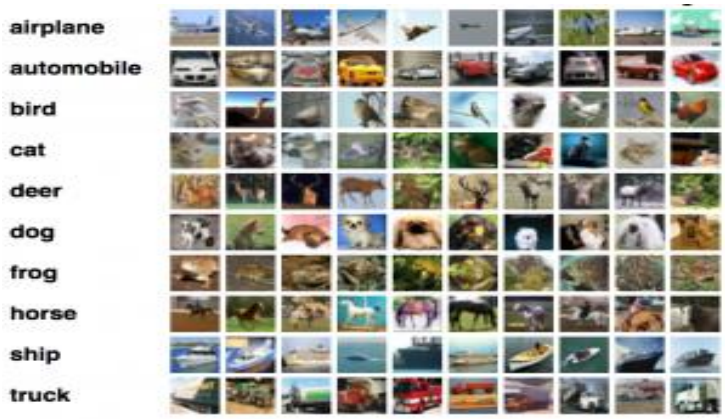
öncelikle değişken tipini ve ele aldığı problemi doğru tanımlamalı ve kodlamasını bu tanımlamaya göre yapmalıdır. Ardından algoritmanın girdilerinden olan uygunluk fonksiyonu (fitness) tanımlanır ve optimize edilmesi gereken amaç fonksiyonu bu fonksiyondur. Geçiş ve Mutasyon gibi genetik operatörler, evrim sürecinin birçok aşamasında stokastik olarak uygulanmaktadır, bu yüzden gerçekleşme olasılıkları belirlenmelidir. Son olarak yakınsama kriterleri sağlanmalıdır ve optimal maliyet ile problem çözülmelidir. GA, problemin çözülebilmesi için lokal değil, global bir araştırma yapar. Problemi etkileyen çok fazla etken varsa, çözümde Genetik Algoritma kullanılması literatürce önerilmektedir.



Şekil-7 Genetik Algoritma

2.3 VERİ SETİ

CIFAR-10 veri seti, 10 sınıftan ve 3 kanalda 32 x 32 pikselden oluşan 60.000 renkli görüntü içerir. Her sınıf 6,000 resim içermektedir. Eğitim seti 50.000 görüntü içerirken, test setleri 10.000 görüntü içerir.

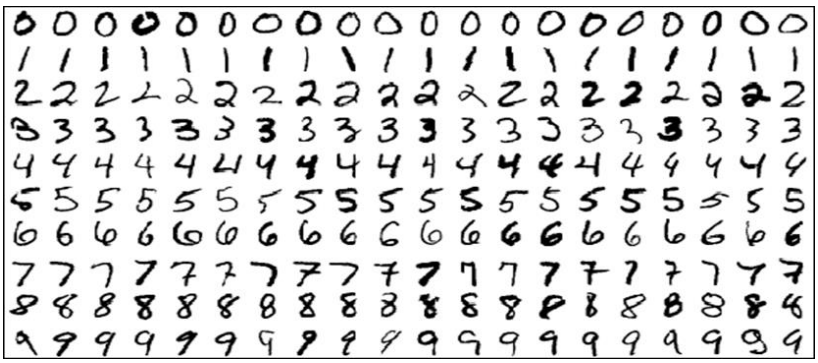


Şekil-7 CIFAR-10 Veri Seti

MNIST (Modified National Institute of Standards and Technology) veri seti kullanılarak sınıflandırma işlemi gerçekleştirilecektir.

Veri seti:

- El yazısı rakamlardan oluşmaktadır.
- Her bir resim 28*28*1 boyutundadır. (Buradaki kanal sayısının 1 olması resimlerin gray-scale olduğunu ifade etmektedir.
- Veri setinde 60.000 eğitim için, 10.000 test için ayrılmış toplam 70.000 adet resim verisi vardır.



Şekil-7 MNIST Veri Seti

3. MODEL OLUŞTURMA

Oluşturulan modelin bütün kısımları adım adım anlatılacaktır:

Öncelikle gerekli import'lar gerçekleştirilmiştir.

```
import keras
import numpy as np
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
```

Keras datasets modülü içerisinde hazır olarak bulunan CIFAR-10 veri seti yüklenmiştir.

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

Veri setimizdeki ground truth (gerçek değerler) one hot encoding işlemine tabi tutulmuş, input verileri reshape işlemi ile vectör haline getirilmiştir.

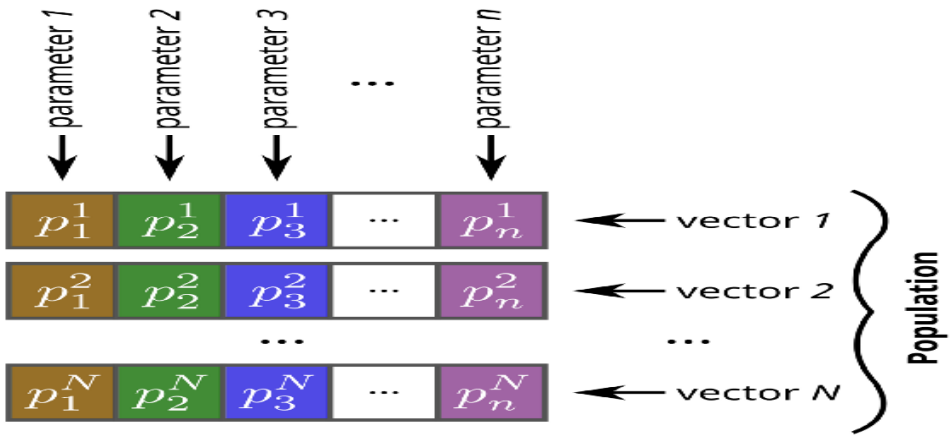
```
input_shape = x_train[1].shape

input_size = 3072

x_train = x_train.reshape(50000, 3072)
x_test = x_test.reshape(10000, 3072)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

Genetik algoritmadaki kromozomlarımızın genlerini oluşturacak parametrelerimizi belirtiyoruz.

```
parameters= [[5,6,7,8,9,10],
             [16,32,64,128,256,512,1024],
             ["tanh","elu","relu","softmax","sigmoid"],
             ["sgd","rmsprop","adam","adagrad","adadelta"],
             ["mean_squared_error","mean_absolute_error","categorical_crossentropy",
              "binary_crossentropy"]
            ]
```



Modelimizi oluşturmak için iki adet sınıf oluşturduk:

- **Network Sınıfı**
- **Genetik Algoritma Sınıfı**

Network Sınıfı

Öncelikle sınıf içerisinde kullanacağımız örnek özniteliklerimizi belirtiyoruz. Parametrelerimizi değişkenlere atayıp rastgele bir hidden layer sayısı belirliyoruz.

```
class Network:
    def __init__(self,input_shape,classes,DNA_parameters,epochs):

        self.architecture = []
        self.fitness = []
        self.acc_history = []
        self.input_shape = input_shape
        self.classes = classes
        self.epochs = epochs

        depth = DNA_parameters[0]
        neurons_per_layer = DNA_parameters[1]
        activations = DNA_parameters[2]
        optimizer = DNA_parameters[3]
        losses = DNA_parameters[4]

        model = Sequential()

        network_depth = np.random.choice(depth)
        self.architecture.append(network_depth)
```

Modelimizde rastgele seçilen derinliğe göre sırasıyla input layer, outpur layer ve hidden layer'ların rastgele nöron sayıları ve aktivasyon fonksiyonları belirlenmiştir. İşlem kolaylığı olsun diye output layer'ın aktivasyon fonksiyonu softmax olarak belirlenmiştir.

Daha sonra her bir model için rastgele şekilde loss fonksiyonu ve optimizer fonksiyonları seçilmiş ardından model compile tanımlanmıştır. Elde edilen her parametre model summary oluşturmak için architecture örnek özniteliği içerisinde saklanmıştır.

```
for i in range(network_depth):
    if i == 0 :
        neurons = np.random.choice(neurons_per_layer)
        activation = np.random.choice(activations)
        self.architecture.append([neurons,activation])
        model.add(Dense(neurons, input_shape = (self.input_shape,),
                        activation = activation))

    if i == network_depth - 1:
        """
        activation = np.random.choice(activations)
        self.architecture.append(activation)
        """
        model.add(Dense(self.classes, activation = 'softmax'))

    else :
        neurons = np.random.choice(neurons_per_layer)
        activation = np.random.choice(activations)
        self.architecture.append([neurons,activation])
        model.add(Dense(neurons, activation = activation))

    model.add(Dropout(0.2))
```

Başlangıçta rastgele oluşturduğumuz popülasyondaki fitness değeri yüksek olan children'ların tekrar üretilme şansları daha yüksek oluyor. Reproduction aşamasından sonra yüksek değerli kromozomlara göre tekrardan children'lar üretmek için fonksiyon tanımlıyoruz.

```
def create_children(self, children):
    model = Sequential()

    children_depth = children[0]

    for i in range(children_depth):
        if i == 0:
            model.add(Dense(children[1][0], input_shape= (self.input_shape,), activation =
                            children[1][1]))

        if i == children_depth - 1:
            model.add(Dense(self.classes, activation = 'softmax'))

        else:
            if i != children_depth - 1:
                model.add(Dense(children[i+1][0], activation = children[i+1][1]))
    model.compile(loss= children[-1][0], optimizer= children[-1][1], metrics=["accuracy"])
    self.model = model
    self.architecture = children
```

Modelimizin fitness fonksiyonunu belirtiyoruz. Modelimizin eğitimini ve testini gerçekleştiriyoruz. Test işlemi sonunda elde edilen her kromozomun Accuracy değerleri fitness değeri olarak kullanılacak. Keras API multiclass için accuracy hesaplarken tahmin edilen değer ile one-hot labelların ne sıklıkla eşleştiğini hesaplıyor.

```
def give_fitness(self):
    return self.fitness

def train(self):
    self.model.fit(x_train, y_train, batch_size = 32, epochs = self.epochs, verbose = 1,
                   shuffle =True)

def test(self):
    loss,acc = self.model.evaluate(x_test, y_test)
    self.fitness = acc
    self.acc_history.append(acc)

def give_DNA(self):
    return self.architecture

def architecture(self):
    self.model.summary()
```

Genetik Algoritma Sınıfı

Öncelikle population_size, mutation_rate, generation ve epochs örnek özniteliklerini belirtip default değerlerini veriyoruz. Populasyon içindeki individualları tutmak için, accuracy değerlerini tutmak için ve her populasyonun ortalama accuracy değerlerini tutmak için boş liste oluşturuyoruz.

```
class GeneticAlgorithm:
    def __init__(self, population_size, mutation_rate, generations = 50, Epochs = 2):
        self.population_size = population_size
        self.mutation_rate = mutation_rate
        self.generations = generations
        self.training_epochs = Epochs
        self.population = None
        self.children_population = []
        self.acces = []
        self.norm_acces = []
```

Population_size kadar bireyden oluşan populasyonumuzu oluşturuyoruz.

```
    def create_population(self):
        self.population = [Network(input_size, num_classes, parameters, self.training_epochs)
                           for i in range(self.population_size)]
```

Popülasyondaki her bir bireyi eğitime ve teste sokup test sonucunda elde edilen accuracy değerini o birey için fitness değeri olarak tutuyoruz.

```
    def train_generation(self):
        for individual in self.population:
            individual.train()

    def predict(self):
        for individual in self.population:
            individual.test()
            self.acc.append(individual.give_fitness())
```

Her populasyonun ortalama accuracy değerlerini tutmak için metot tanımlıyoruz.

```
def normalize(self):
    sum_ = sum(self.acc)
    self.norm_acc = [i/sum_ for i in self.acc]
```

Her generasyondan sonra acc ve norm_acc değerlerini temizlemek için metot tanımlıyoruz.

```
def clear_losses(self):
    self.norm_acc = []
    self.acc = []
```

Popülasyondaki her bir bireyi alıp gen sayısına göre eğer rastgele ürettiğimiz sayı mutation_rate'den küçük ise mutasyona uğratıyoruz. Eğer ilk gen için rastgele üretilen sayı mutasyon oranından küçük ise kromozomdaki katman sayısı geni rastgele olarak seçilip değiştiriliyor.Loss ve Optimizer fonksiyonları için ayrı bir döngü tanımlandı eşit oranda. Aktivasyon fonksiyonu ve nöron sayısı için ayrı bir döngü tanımladık.

```
def mutate(self):
    for child in self.children_population:
        for i in range(len(child)):
            if np.random.random() < self.mutation_rate:
                print("\nMutation!")
                if i == 0:
                    new_depth = np.random.choice(parameters[0])
                    child[0] = new_depth


                if i == len(child) - 1:

                    if np.random.random() < 0.5:
                        new_loss = np.random.choice(parameters[4])
                        child[-1][0] = new_loss
                    else :
                        new_optimizer = np.random.choice(parameters[3])
                        child[-1][1] = new_optimizer

                if i != 0  and i != len(child)-1:

                    if np.random.random() < 0.33:
                        new_activation = np.random.choice(parameters[2])

                        child[i][1] = new_activation

                    else:
                        new_neuron_count = np.random.choice(parameters[1])
                        child[i][0] = new_neuron_count
```

Midpoint-crossover kullanıldı. İlk önce fitness değerine göre parent seçimi yapıyoruz. Eğer iyi parent bulamazsak rastgele parent seçiyoruz. Crossover için rastgele midpoint seçiyoruz. Yeni birey üretip eski popülasyondakini onla değiştiriyoruz.

```
def crossover(self):

    population_idx = [i for i in range(len(self.population))]
    for i in range(len(self.population)):

        if sum(self.norm_acc) != 0:
            parent1 = np.random.choice(population_idx, p = self.norm_acc)
            parent2 = np.random.choice(population_idx, p = self.norm_acc)

        else:
            parent1 = np.random.choice(population_idx)
            parent2 = np.random.choice(population_idx)

        parent1_DNA = self.population[parent1].give_DNA()
        parent2_DNA = self.population[parent2].give_DNA()

        mid_point_1 = np.random.choice([i for i in range(2,len(parent1_DNA)-2)])
        mid_point_2 = np.random.choice([i for i in range(2,len(parent2_DNA)-2)])

        child = parent1_DNA[:mid_point_1] + parent2_DNA[mid_point_2:]
        new_nn_depth = len(child)-2
        child[0] = new_nn_depth
        self.children_population.append(child)

    self.mutate()
    keras.backend.clear_session()
    for i in range(len(self.population)):
        self.population[i].create_children(self.children_population[i])
```

Son generation hariç diğer generation'larda crossover yaparak evolution işlemi yaptırıyoruz. Her generation'daki popülasyonların accuracy değerlerini çizdiriyoruz.

```
def run_evolution(self):
    for episode in range(self.generations):
        print("\n--- Generation  {} ---".format(episode))
        self.clear_losses()
        self.train_generation()
        self.predict()
        if episode != self.generations - 1:
            self.normalize()
            self.crossover()

        else:
            pass
        self.children_population = []

    for a in range(self.generations):
        for member in self.population:
            plt.plot(member.acc_history)
    plt.xlabel("Generations")
    plt.ylabel("Accuracy")
    plt.show()
```

Modelimizi çalıştırıyoruz.

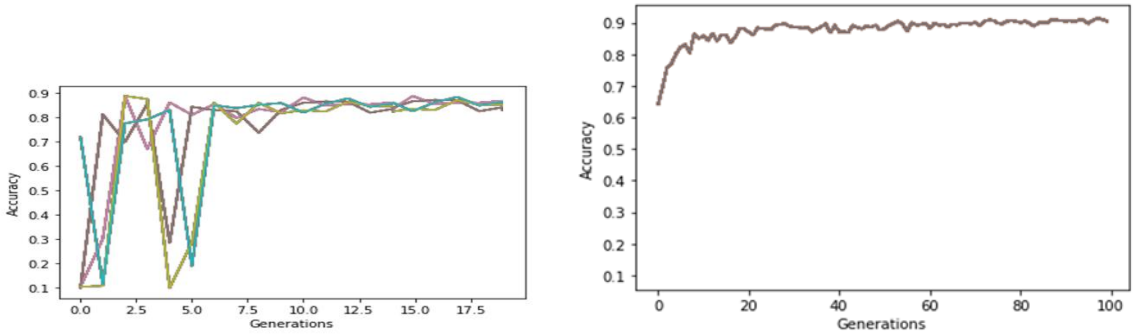
```
GA = GeneticAlgorithm(population_size= 4, mutation_rate= 0.03, generations= 100, Epochs= 1)
GA.create_population()
GA.run_evolution()
```

4. SONUÇLAR VE GELECEK ÇALIŞMA

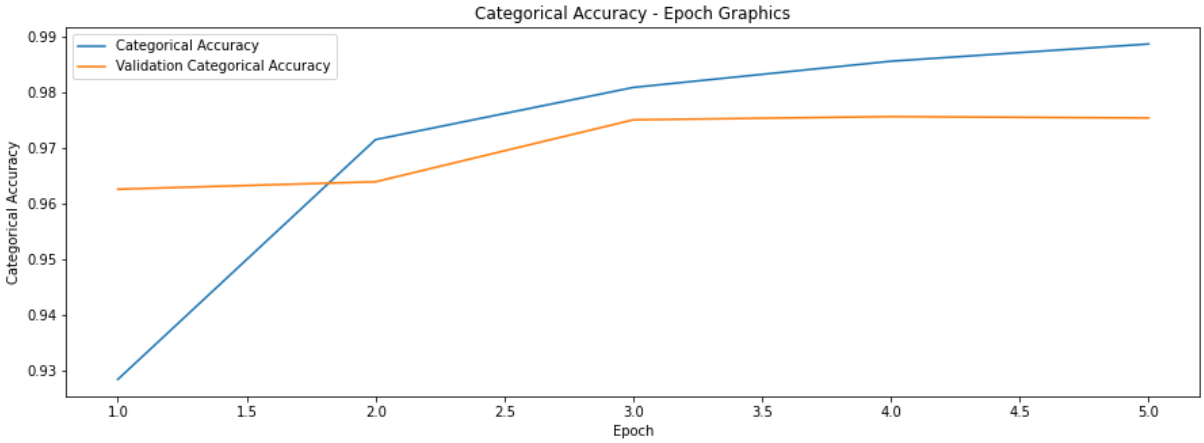
MNIST Veri Seti İçin:

Modelimiz 15 epoch ve 100 epoch'lık eğitimler sonunda %90 accuracy değerlerine ulaşmıştır. Elde edilen sonuçlar sezgisel olarak oluşturduğumuz fully connected model ile karşılaştırılmıştır. Sonuçta grafikler üzerinden de görüleceği gibi genetik algoritma modelimiz daha kötü sonuç vermiştir (GA algoritma-→%90, Sezgisel model-→%98). Aynı şekilde doyuma ulaşan epoch değerleri karşılaştırıldığında yaklaşık aynı oldukları görülmektedir. Özellikle genetik algoritma modelinin mutasyon ve crossover metotlarında gidilecek değişikliklerle modelimizin sonuçları iyileştirilebilir.

GA Algoritma (MNIST) Accuracy Eğrisi



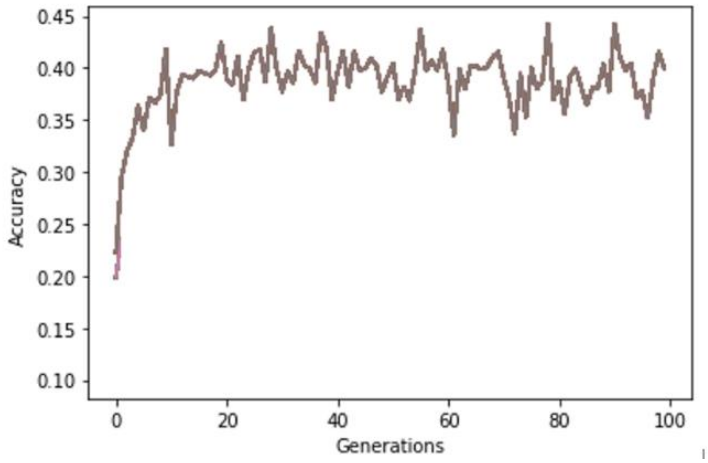
Sezgisel Model (MNIST) Accuracy Eğrisi



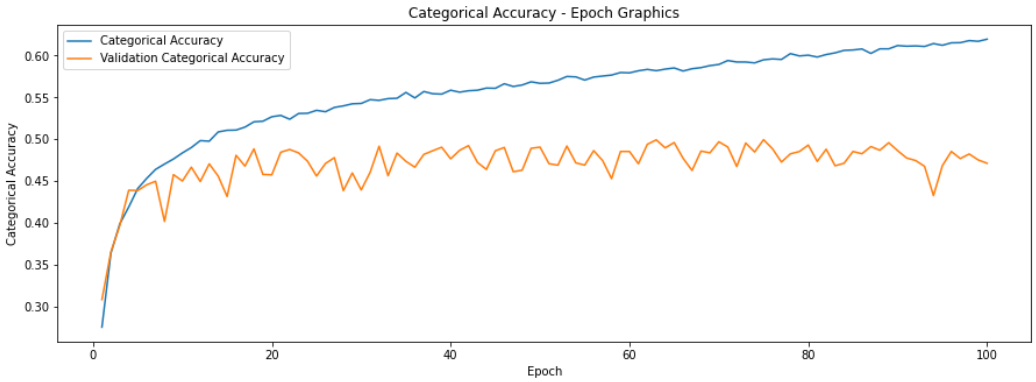
CIFAR-10 Veri Seti İçin:

Modellerimizi CIFAR-10 veri seti ile 100'er epoch eğitim sonunda karşılaştırdığımızda modellerin yaklaşık aynı sonucu verdiği grafikler üzerinde görülmektedir (GA algoritma-→%40, Sezgisel model-→%45). Bu veri setinde modellerin daha başarısız sonuçlar vermesinin sebebi dense layer'ların veri içindeki ayırt edici özellikleri yakalamada başarısız olmasından kaynaklanmaktadır. Bu veri setinin convolutional layer'lar üzerinden eğitilmesi daha doğru sonuçlara ulaşmamızı sağlayacaktır.

GA Algoritma (CIFAR-10) Accuracy Eğrisi



Sezgisel Model (CIFAR-10) Accuracy Eğrisi



Gelecek çalışma olarak kullanılan genetik algoritmada parametre olarak learning rate eklenmesi ve diğer derin öğrenme modelleri (CNN-

LSTM-BİDİRECTİONAL) üzerinde hiper parametre ayarlamalarının yapılması planlanmaktadır.

KAYNAKLAR

1-) Genetic Algorithms (Xin-She Yang, in Nature-Inspired Optimization Algorithms (Second Edition)

2-)https://ai.googleblog.com/2017/05/using-machine-learning-to-explore.html

3-) E. Wirsansky, Hands-on genetic algorithms with Python: applying genetic algorithms to solve real-world deep learning and artificial intelligence problems.

4-) S. Lee, J. Kim, H. Kang, D. Y. Kang, and J. Park, "Genetic algorithm based deep learning neural network structure and hyperparameter optimization," Applied Sciences (Switzerland), vol. 11, no. 2, pp. 1–12, Jan. 2021, doi: 10.3390/app11020744.

5-) F. David Krüger and M. Nabeel, "Hyperparameter Tuning Using Genetic Algorithms A study of genetic algorithms impact and performance for optimization of ML algorithms."

6-) M. Tayebi and S. el Kafhali, "Hyperparameter Optimization Using Genetic Algorithms to Detect Frauds Transactions," 2021, pp. 288–297. doi: 10.1007/978-3-030-76346-6_27.

7-) H. Chung and K. S. Shin, "Genetic algorithm-optimized long short-term memory network for stock market prediction," Sustainability (Switzerland), vol. 10, no. 10, Oct. 2018, doi: 10.3390/su10103765.

8-) F. David Krüger and M. Nabeel, "Hyperparameter Tuning Using Genetic Algorithms A study of genetic algorithms impact and performance for optimization of ML algorithms."