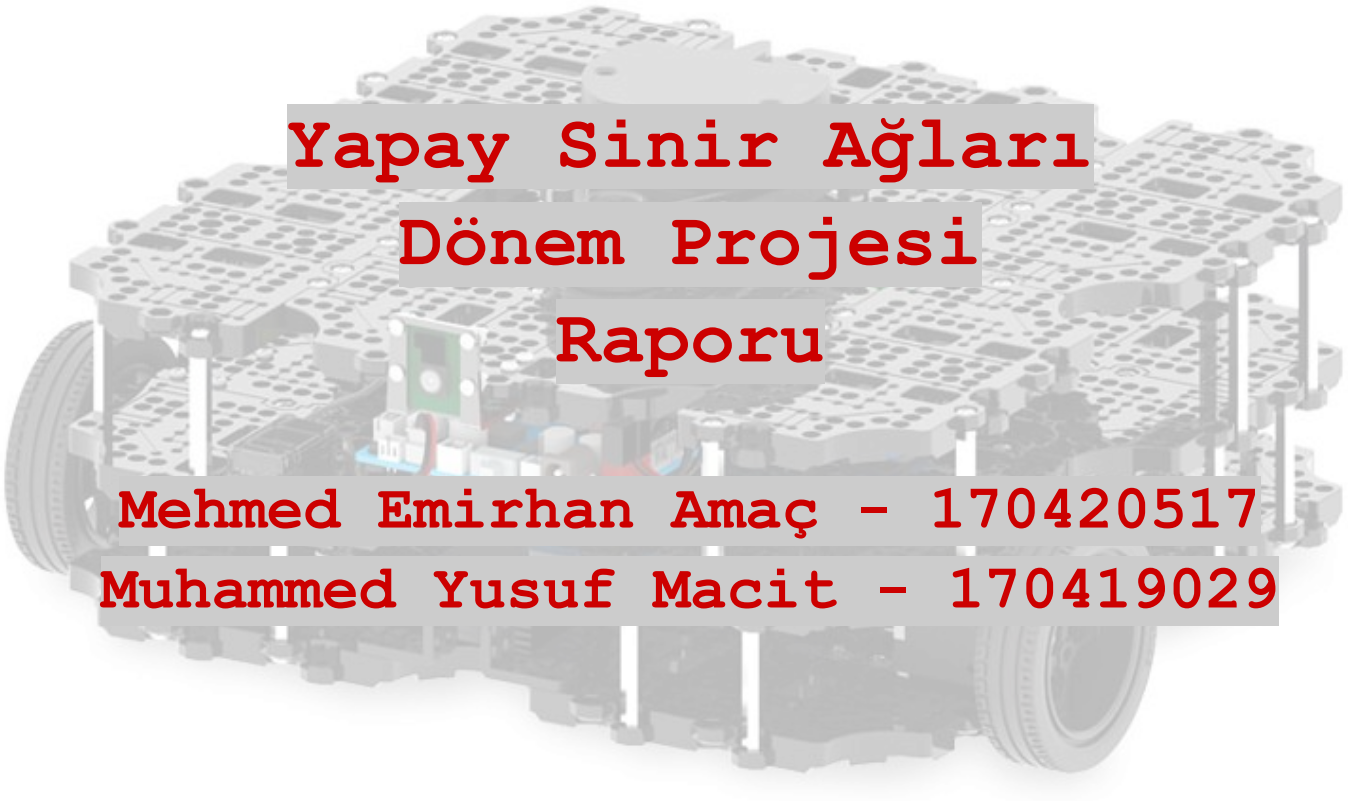


Yapay Sinir Ağları
Dönem Projesi
Raporu

Mehmed Emirhan Amaç - 170420517
Muhammed Yusuf Macit - 170419029

Proje Github Linki:https://github.com/amacemirhan/YSAProje
Proje dataset Linki: Arrow Direction Detection

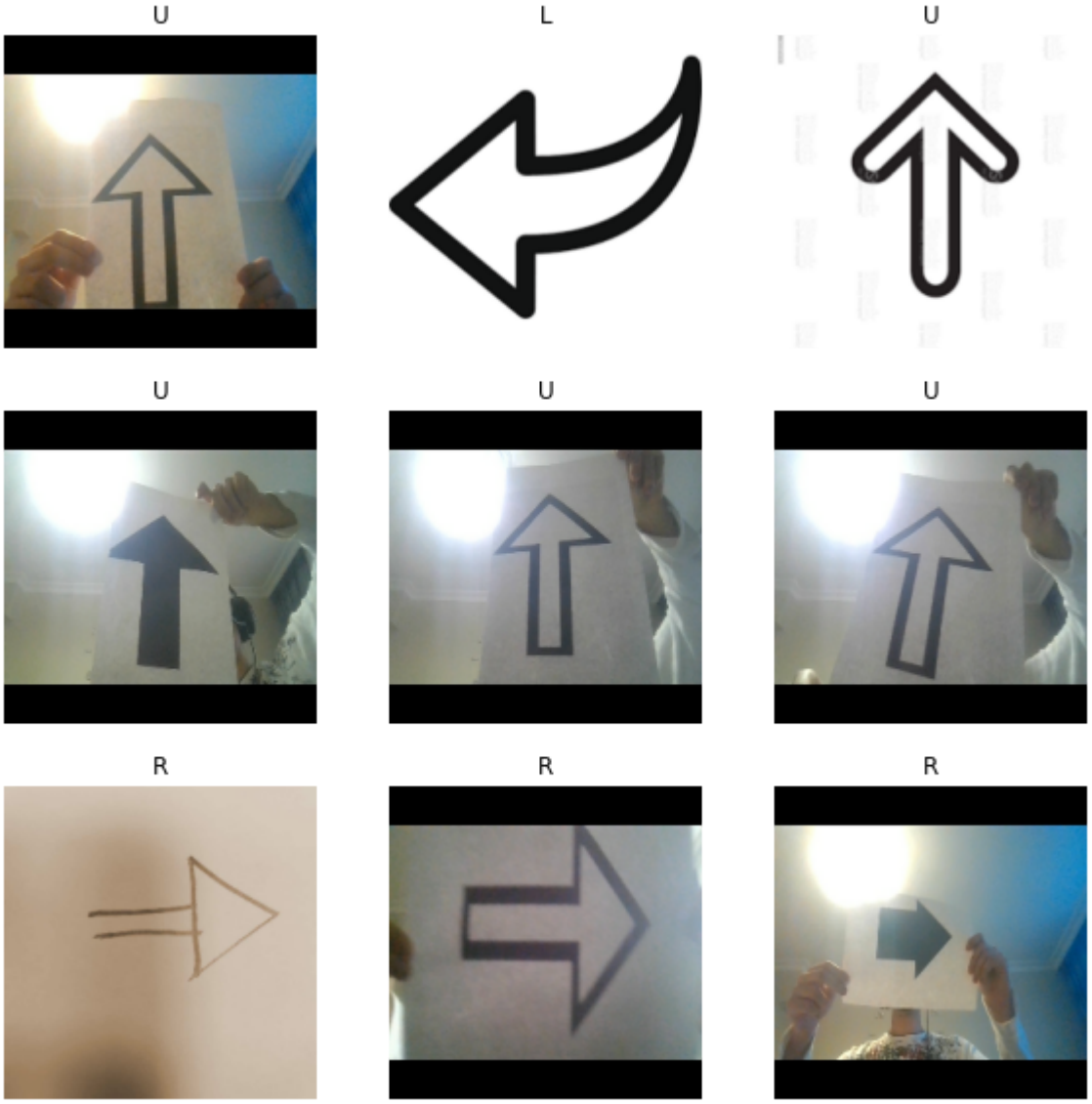
Dataset Oluşturma

Datasetimizi biz kendimiz oluşturmak istedik.Okulda buluşup banknot kağıtlara oklar çizdik ve bu okların telefon kamerasından görüntüsünü aldık sağ, sol, yukarı olarak üç ayrı klasöre atarak oklarımızı etiketledik. Çektiğimiz görüntü sayısının yetersiz olduğunu düşündük.Google görsellerden dijital ok resimleri ayıkladık.

MANUEL DATA AUGMENTATION

Elimizdeki veriyi arttırmak için öncelikle çektiğimiz okları 90 derece döndürüp birbirlerine kattık. Daha sonra Adobe Photoshop uygulamasından fotoğraflara topluca *sepya tonu* ekledik.En sonunda da hala yetersiz hissettiğimizden ve realtime için daha uygun olacağını düşündüğümüzden bilgisayar kamerasından seri fotoğraf çekme scripti yazdık. Github linkinde bulabilirsiniz scriptin adı WebCamSnapshot.py
Bu script ile kağıda çizdiğimiz okun 0.7 saniyede bir 40 arlı fotoğraflarını çektik. Çekim sırasında kağıdı hareket ettirdiğimiz için her fotoğraf diğerinden farklı oldu.Uygun klasörleyip datasetimize ekledik ve en son train datasetimizi 1741 tane fotoğrafa çıkarttık. Test datasetimiz de 31 adet.Bunun az olmasının sebebi bizim asıl

realtime’a yönelik çalışmamız ve testlerimizi realtime yapmamızdır. Bu veri artırımının sonucu daha uzun eğitim süresi ve daha doğru realtime sonuçlar olarak bize geri yansıdı.

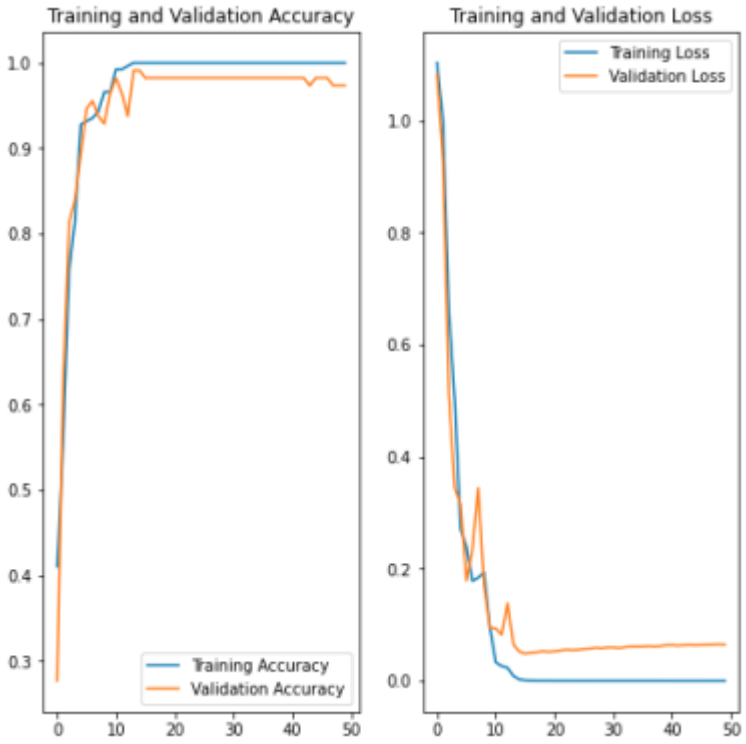


Model Eğitimi Ve Sonuçlar

İlk modelimizi tensorflow’un sitesinden alıp denedik. Model şu şekilde 10 adet katmandan oluşuyor.

```
num_classes = len(class_names)

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

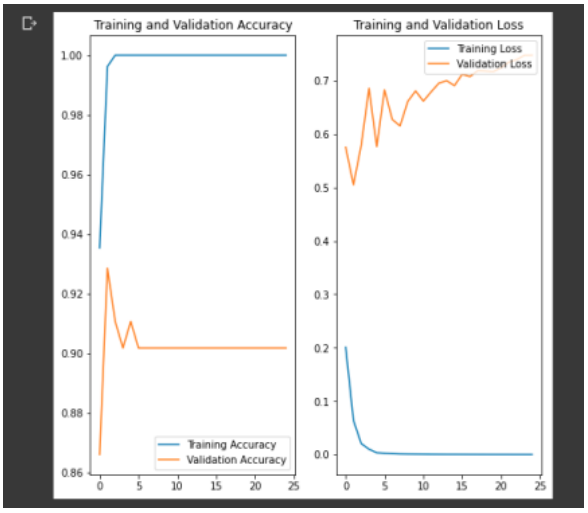
50 epoch eğitimden aşağıdaki gibi bir grafik oluştu.

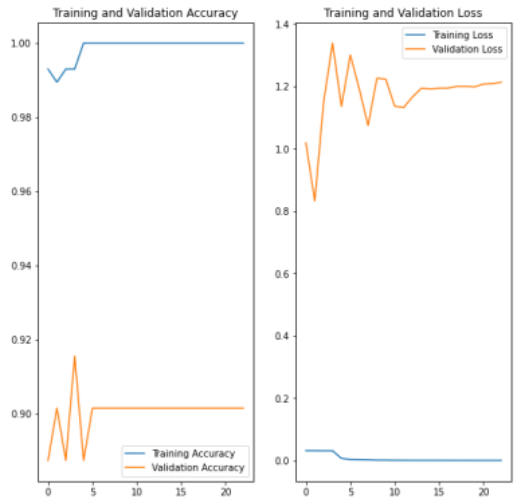


Bu modeli oluşturduğumuz 30 adetlik train de kullanılmayan fotoğraflardan oluşan test datasetinde deneyince %96.7 lik bir başarıya ulaştı.

```
tahmin:  Yuk     cevap:  Yuk
tahmin:  Sol     cevap:  Sağ
tahmin:  Sol     cevap:  Sol
tahmin:  Sağ     cevap:  Sağ
tahmin:  Sol     cevap:  Sol
tahmin:  Sağ     cevap:  Sağ
tahmin:  Sol     cevap:  Sol
tahmin:  Sol     cevap:  Sol
tahmin:  Yuk     cevap:  Yuk
tahmin:  Sağ     cevap:  Sağ
tahmin:  Sağ     cevap:  Sağ
tahmin:  Sol     cevap:  Sol
tahmin:  Sağ     cevap:  Sağ
tahmin:  Yuk     cevap:  Yuk
tahmin:  Sağ     cevap:  Sağ
tahmin:  Sol     cevap:  Sol
tahmin:  Sağ     cevap:  Sağ
tahmin:  Sağ     cevap:  Sağ
tahmin:  Yuk     cevap:  Yuk
tahmin:  Sol     cevap:  Sol
96.7741935483871
```

Daha sonra sırf meraktan araya Conv2D ve Pooling katmanları ekledik ve modelin öğrenemediğini gördük. Sonuçlar aşağıdaki gibi oldu.



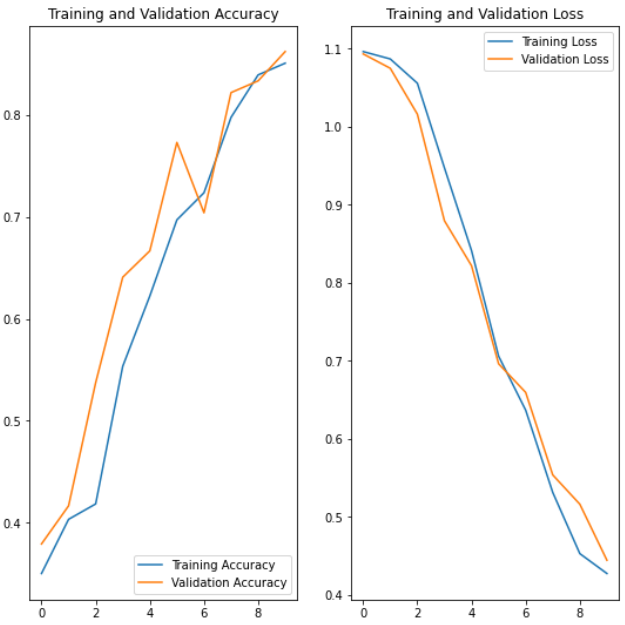


Kendi Modelimizi Denedik

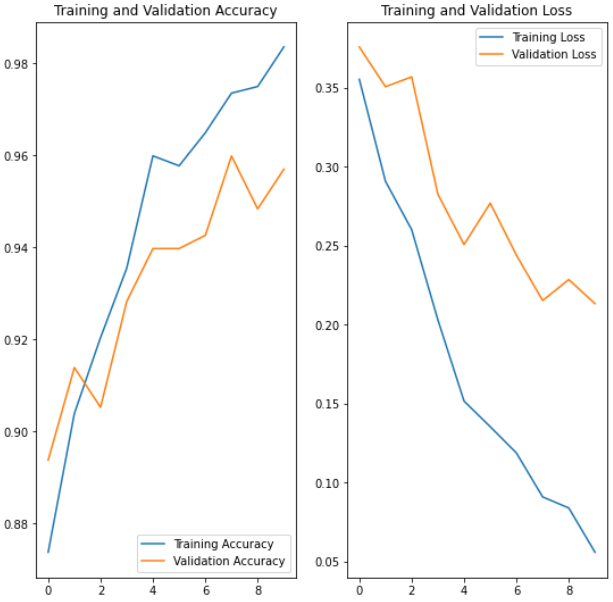
```
[162] model = Sequential([
        layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
        layers.Conv2D(16, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(16, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(32, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(32, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(64, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(64, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(64, 3, padding='same', activation='relu'),
        layers.MaxPooling2D(),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(128, activation='relu'),
        layers.Dense(128, activation='relu'),
        layers.Dense(3,activation='softmax')
      ])
```

```
[163] model.compile(optimizer='Adamax',
                    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                    metrics=['accuracy'])
```

Hem hızlı hem de doğru sonuç veren model oluşturabilmek için birçok deneme yaptık ve en son Yukarıda görseli verilen modeli kullanmaya karar verdik. Modelimizde 138.387 parametre bulunmaktadır. Bu sayede diğer modellere kıyasla çok daha hızlı öğrenme gerçekleştiriyor.

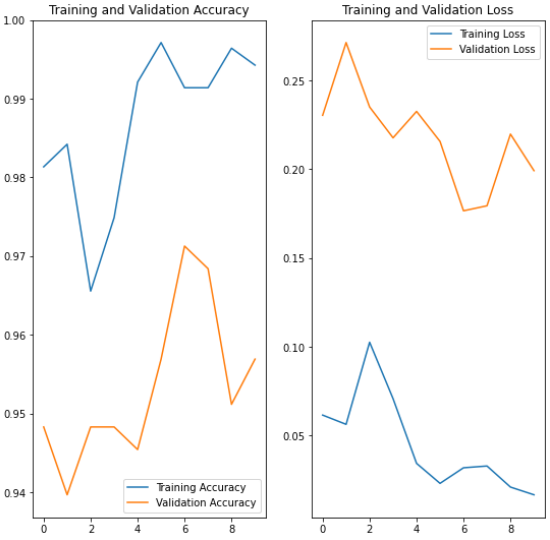


Yukarıda görüldüğü üzere optimizer olarak Adamax kullandık. En sondaki Dense layer da softmax activation fonksiyonu kullandık. Bu modeli ilk önce 10 epoch eğittik.



Daha sonra overfit olmadığını gördük ve aynı ağırlıkların üzerine 10 epoch daha eğittik.Son değerler şu şekilde oldu:

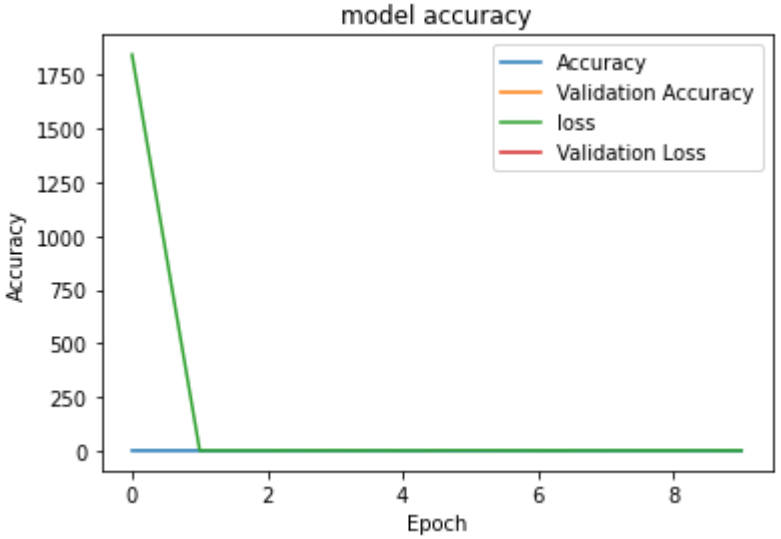
```
loss: 0.0557 - accuracy: 0.9835 - val_loss: 0.2132 - val_accuracy: 0.9569
```



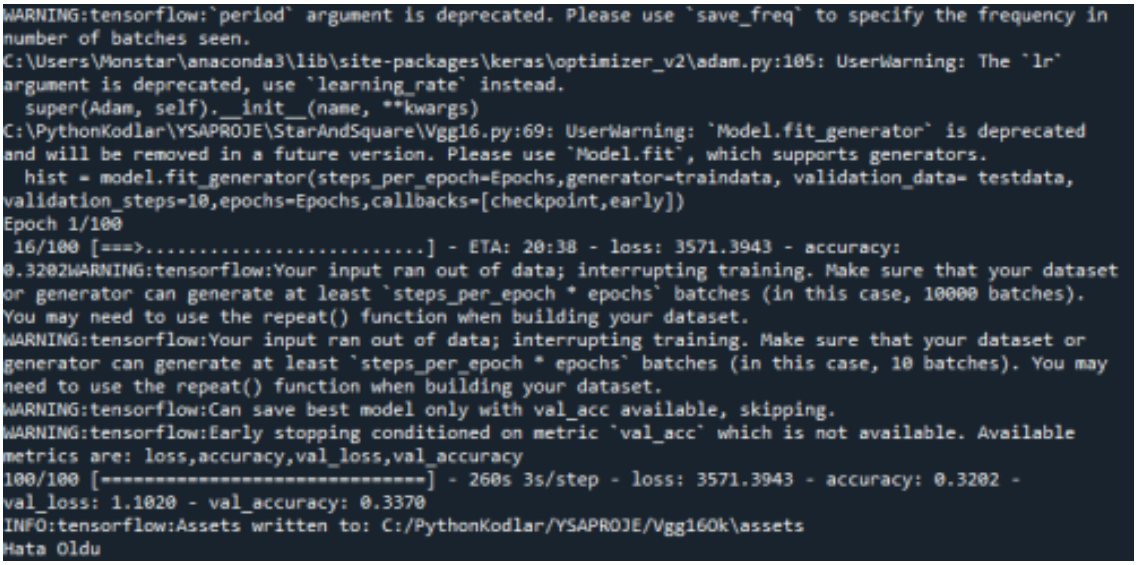
10 epoch daha eğittikten sonra yavaştan doyum noktasına ulaştığını gözlemledik.10 ar epoch aralıklarla eğitmemizin sebebi her 10 epochta bir save alıp eğer son eğitim overfit olursa onu almayıp tüm eğitimin boşa gitmemesini istememizdir.
10 epoch daha eğitip gördük ki daha fazla öğrenemiyor. Validation loss en fazla 0.18 civarlarına kadar düşürebildik.

VGG 16 Modeli Denemeleri

VGG 16 modelini denedik çok ağır ve büyük bir model olduğunu gördük ilk eğitimi 10 Epoch 1700 lük fotoğraf verisiyle eğittik 39 dk sürdü çok birşey kat edemedi.
İkinci deneme:



val_acc parametresinin yanlış girilmesinden dolayı grafik hatalı çıkmıştır.



VGG 16 modelini 100 epoch eğitmek için çalıştırdık ancak 1. epochun 16. batchini işlerken hata vermiştir. Bunun nedeni eğitimde girilen parametrelerden kaynaklanıyordu.

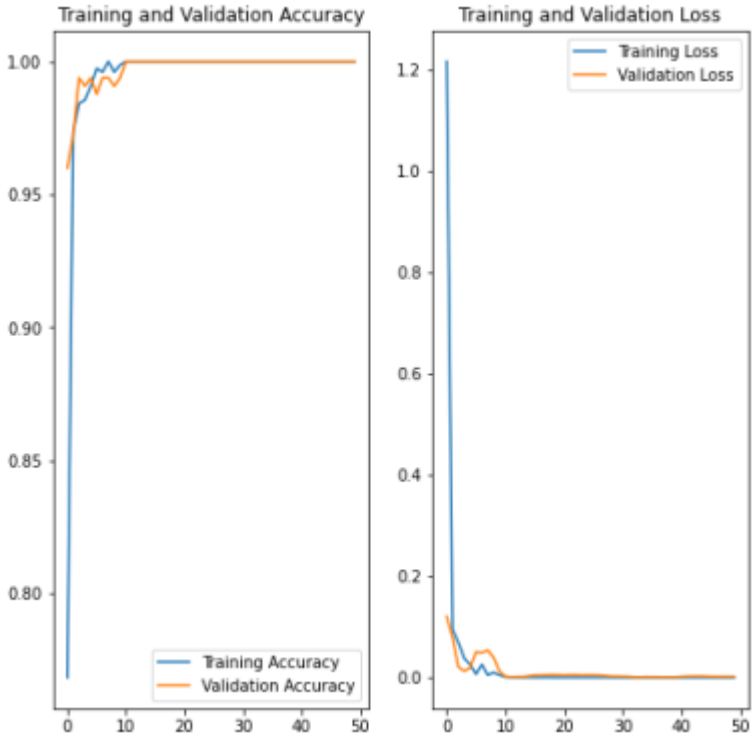
20 Epochluk Vgg16 denemesinden sonra Realtime olarak Oklarla model test edilmiştir. Model karşısına çıkan her şeye Sol ok cevabını vermiştir

Vgg16 modelini farklı zamanlarda birçok kez eğitmeyi denedik ama öğrenmesi çok ağır kalıyordu. 27. epochta daha doğru düzgün öğrenmeden Early Stoppinge giriyor accuracy 0.3594, val_accuracy:0:3469 da kalıyordu. Biz de bu yüzden Vgg16 modelini eğitmekten vazgeçtik.

EfficientNetB0 Denemeleri

İnternette makale okurken Image Classificationda modellerin parametre ve başarı oranlarını karşılaştıran aşağıdaki tabloyu gördüm. Parametre azlığına rağmen diğerlerinin başarısına yakın başarı gösterebilen EfffficientNetB0 modelinin denemeye değer olduğunu düşündük.

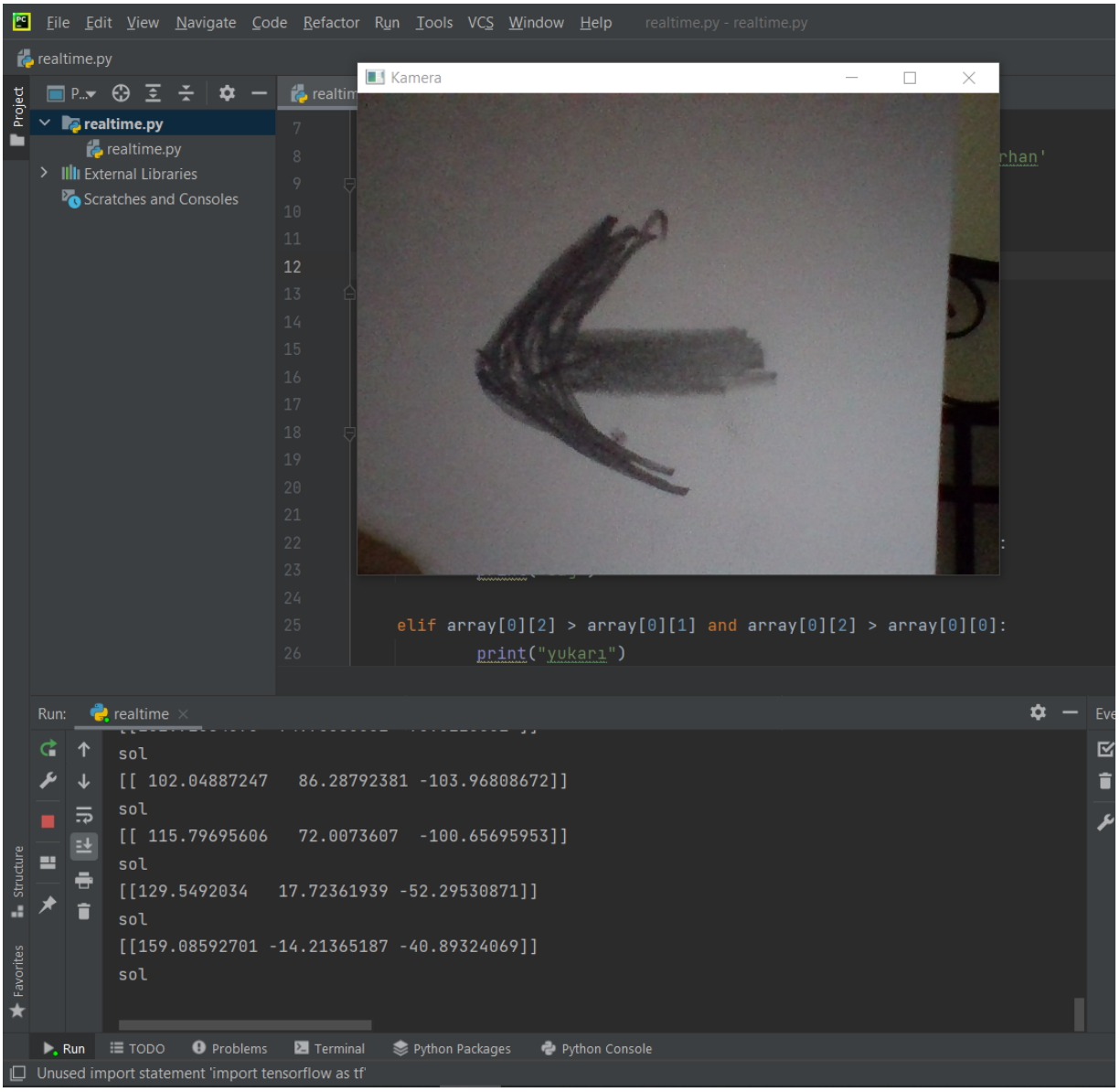
Model	Year	Number of Parameters	Top-1 Accuracy
VGG-16	2014	138 Million	74.5%
ResNet-50	2015	25 Million	77.15%
Inception V3	2015	24 Million	78.8%
EfficientNetB0	2019	5.3 Million	76.3%
EfficientNetB7	2019	66 Million	84.4%



Grafikten görüldüğü üzere 50 epoch eğitimin sonucu ciddi oranda başarı sağladı.

Realtime Denemeleri

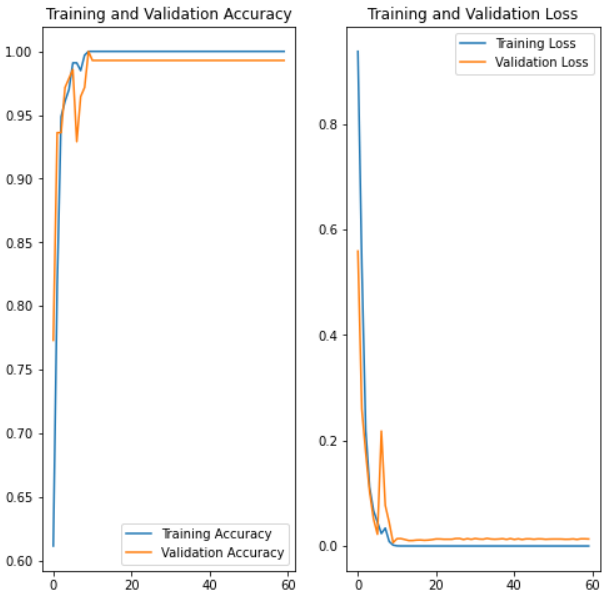
İlk eğittiğimiz modeli realtime denedik. Ancak başarı oranı epey düştü. 3 te 1 ini anca bilebiliyordu. Daha sonra bilgisayar kamerası ile genişlettiğimiz ok veri setini kendi modelimizle eğitip denediğimizde idare eder bir başarı oranı sağladık. Aşağıda da realtime denemeden alınmış ekran görüntüsü bulunuyor.



Kendi datasetimizi kullanarak eğittiğimiz EfficientNetB0 modeli realtimeda neredeyse kusursuz bir başarı sağladı.Test Videosu github linkinde EfficientNetB0 Realtime Test.mp4

Yan Proje

Oklarla eğittiğimiz ilk tensorflow modelinde verileri tek tek verdiğimizde modelimizin gayet iyiydi fakat realtime çalışırken beklentimizin altında kaldı. Sınıflandırma yaparken okun yönünü tayin etmesinin daha zor olabileceğini bunun yerine farklı şekiller ile eğiterek daha iyi sonuç alacağımızı düşündük. Bunun için Yıldız, Kare ve Ok olmak üzere 3 sınıflı bir veri seti daha hazırladık. Hem bu veri seti ile hem de diğer veri seti ile denemeler yaptık.



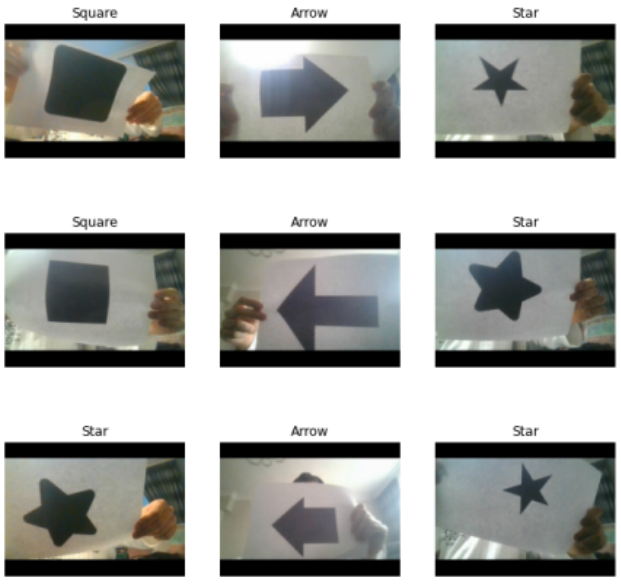

```
model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Yıldız, kare ve ok kullandığımız modelin katmanları ve Accuracy/Loss Grafikleri.

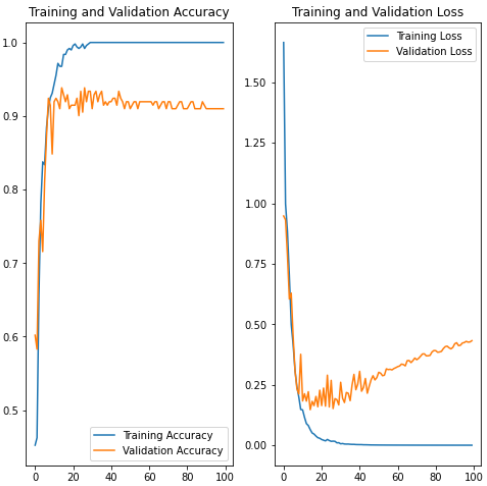
Bu projede kullandığımız veri setine aşağıdaki linkten ulaşabilirsiniz.
[Dataset2]

Yukarıda kendi oluşturduğumuz modeli ve kendi oluşturduğumuz veri setini kullandığımız eğitimin realtimedaki başarısı ana projemizin başarısına göre daha iyiydi.
Bunun nedeni olarak burada kullanılan 3 şeklin birbirinden daha kolay ayırt edilebilir olması daha başarılı öğrenme gerçekleşmesini sağladığını düşünüyoruz.

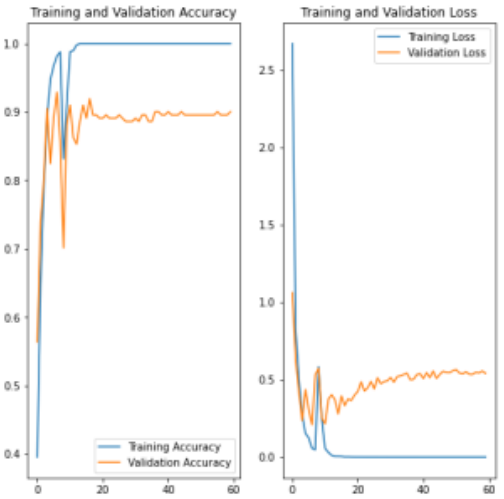
Sadece okların yönlerini ayırt etmede eğittiğimiz modellerin validation accuracy'si 0.18 lere kadar düşebiliyorken bu yıldızlı,kareli,oklu öğrenmede bu değer 0.05 lere kadar düşebiliyor.



Yan projemize ait datasetten bazı örnekler.



100 epoch
224x224
batch size 128



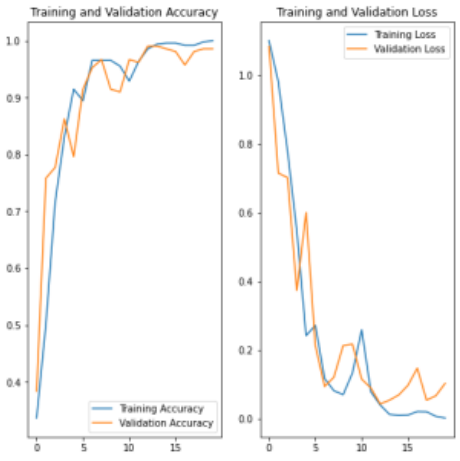
60 epoch
640x480
batch size 16

İki model de overfitting'e girdiği için hatalı sonuç verdi.

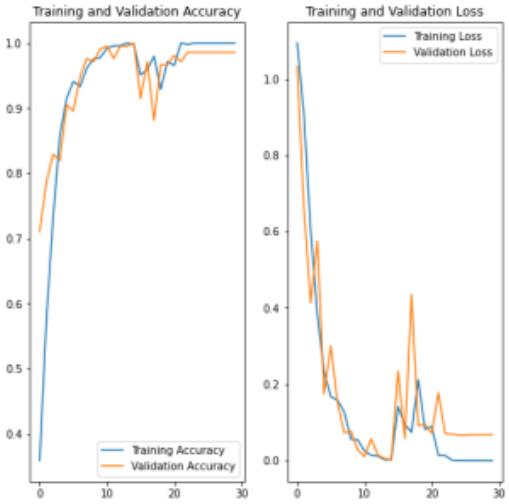
Yeni katmanlar ekleyip input size=640x480, batch size=16 olarak 30 epoch eğittiğimiz model realtime test ettiğimizde beklediğimizden daha iyi sonuç verdi.

```
#Model
model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Modeldeki Katmanlar



Modelin 20 Epochluk eğitiminin grafiği



Modeli 30 Epoch eğittikten sonra Accuracy/Loss Grafikleri.