In [ ]:
```
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten,Dropout, Resizing
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.vgg19 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import Sequential
from tensorflow.data import AUTOTUNE, Dataset
from tensorflow.python.client import device_lib
# from tensorflow.data.Dataset import from_tensor_slices
from tensorflow import cast, float32, expand_dims, TensorSpec
import tensorflow as tf
import numpy as np
##import pandas as pd
import os
import cv2
import matplotlib.pyplot as plt
```

Data behandling

Den første del går ud på at klargøre vores data til brug.
Her starter vi med at skrive path op til vores trænings, test og valideringsdata.

In [ ]:
```
train_path="data/train"
test_path="data/test"
val_path="data/valid"
```

Dem vil vi så bruge til at hente alle billederne.
De bliver gemt i en liste og bliver fundet ved at gå igennem alle subfolders:

```
for folder in os.listdir(train_path):
  sub_path=train_path+"/"+folder
  for img in os.listdir(sub_path):
```

Her kan man se at vi iterator over alle directories i trainingsmappen.
Der tager vi så alle de billeder og loader dem med cv2 samt ændre størrelsen på billederne så de kræver mindre at bruge.

```
image_path=sub_path+"/"+img
img_arr=cv2.imread(image_path)
img_arr=cv2.resize(img_arr,(224,224))
train_x.append(img_arr)
```

In [ ]:
```
print("Retrieving training Data")

train_x=[]
for folder in os.listdir(train_path):
  sub_path=train_path+"/"+folder
  for img in os.listdir(sub_path):
    image_path=sub_path+"/"+img
```

```
    img_arr=cv2.imread(image_path)
    img_arr=cv2.resize(img_arr,(224,224))
    train_x.append(img_arr)

print("Retrieving test Data")

test_x=[]
for folder in os.listdir(test_path):
  sub_path=test_path+"/"+folder
  for img in os.listdir(sub_path):
    image_path=sub_path+"/"+img
    img_arr=cv2.imread(image_path)
    img_arr=cv2.resize(img_arr,(224,224))
    test_x.append(img_arr)

print("Retrieving validation Data")

val_x=[]
for folder in os.listdir(val_path):
  sub_path=val_path+"/"+folder
  for img in os.listdir(sub_path):
    image_path = sub_path+"/"+img
    img_arr=cv2.imread(image_path)
    img_arr=cv2.resize(img_arr,(224,224))
    val_x.append(img_arr)
```

```
Retrieving training Data
Retrieving test Data
Retrieving validation Data
```

Efter det konverterer vi dataen til et numpy array, så vi kan arbejde med det.
Det bruger vi så til at broadcaste en division på alle elementer i arrayet, så vi får en værdi mellem 0 og 1.

In [ ]:

```
train_x=np.array(train_x)
test_x=np.array(test_x)
val_x=np.array(val_x)
train_x=train_x/255.0
test_x=test_x/255.0
val_x=val_x/255.0
```

Den næste del af koden finder labels til vores billeder så vi senere kan bruge dem til at træne.
Det gør vi ved hjælp af `ImageDataGenerator` man kan bruge i sammenarbejde med `flow_from_directory()`.

In [ ]:

```
train_datagen = ImageDataGenerator(rescale = 1./255)
test_datagen = ImageDataGenerator(rescale = 1./255)
val_datagen = ImageDataGenerator(rescale = 1./255)
training_set = train_datagen.flow_from_directory(train_path,
 target_size = (224, 224),
 batch_size = 32,
 class_mode = 'sparse')
test_set = test_datagen.flow_from_directory(test_path,
```

```
 target_size = (224, 224),
 batch_size = 32,
 class_mode = 'sparse')
val_set = val_datagen.flow_from_directory(val_path,
 target_size = (224, 224),
 batch_size = 32,
 class_mode = 'sparse')
```

```
Found 9700 images belonging to 5 classes.
Found 100 images belonging to 5 classes.
Found 200 images belonging to 5 classes.
```

Dem kan vi så gemme seperat så vi har labels og billeder i henholdsvigst y og x arrays.
Det gøres ved at tage `.classes`:

In [ ]:
```
train_y=training_set.classes
test_y=test_set.classes
val_y=val_set.classes
```

Her kan man se at der er en del mere træningsdata da det er det vi har brug for mest af:

In [ ]:
```
train_y.shape,test_y.shape,val_y.shape
```

Out[ ]:
```
((9700,), (100,), (200,))
```

Modelgeneration

Nu har vi forberedt dataen og kan begynde at genererer modelen.
Her har vi brugt størrelsen på billederne med en ekstra dimension på 3 for hver farvekanal.
Her starter vi med at bruge VGG19 hvor vi definerer størrelsen på inputtet samt hvilken type vægte vi gerne ville bruge.

Vi sætter også alle de lag der følger med VGG19 til ikke at skulle trænes da de allerede er fortrænet.

In [ ]:
```
vgg = VGG19(input_shape=[224,224] + [3], weights='imagenet', include_top=False)

for layer in vgg.layers:
  layer.trainable = False
```

Herefter kan vi tilføje til modelen så den passer til vores brug.

Her starter vi med at flatten vgg output. Der kan vi så efter bruge et dense lag hvor vi har specificeret 5 output node hvilket passer da vi har fem typer af frugt.
Her bruger vi også softmax som aktivations classifier da vi arbejder med multi-class klassifikation.

In [ ]:
```
x = Flatten()(vgg.output)
```

```
prediction = Dense(5, activation='softmax')(x)
model = Model(inputs=vgg.input, outputs=prediction)

#For at få et overblik over modellen kan vi:
model.summary()
```

Model: "functional_3"

Layer (type)	Output Shape	Par
input_layer_1 (InputLayer)	(None, 224, 224, 3)	
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2,359
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2,359
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	
flatten_1 (Flatten)	(None, 25088)	
dense_1 (Dense)	(None, 5)	125

Total params: 20,149,829 (76.87 MB)

Trainable params: 125,445 (490.02 KB)

Non-trainable params: 20,024,384 (76.39 MB)

Så kan man compile modellen hvor vi har brugt accuracy som vores metric at måle hvor godt den klarer sig på.

Her sætter vi også early_stop op der siger at hvis den ser at vi mister kvalitet for mange gange i træk stopper den før den er helt færdig.
Det er praktisk da man tit kan risikerer ikke at få noget ud af det sidste og at det derfor bare er spild af tid.

In [ ]:
```
model.compile(
loss='sparse_categorical_crossentropy',
optimizer="adam",
metrics=['accuracy']
)
```

In [ ]:
```
from tensorflow.keras.callbacks import EarlyStopping
early_stop=EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=5)
```

Til sidst kan vi kører modellen hvor vi har specificeret vores early_stop samt trænings- og valideringsdata.

Det gør vi da den bruger valideringsdataen til at vurderer den selv efter hver epoke.

In [ ]:
```
result = model.fit(
  train_x,
  train_y,
  validation_data=(val_x,val_y),
  epochs=10,
  callbacks=[early_stop],
  batch_size=32,shuffle=True)
```

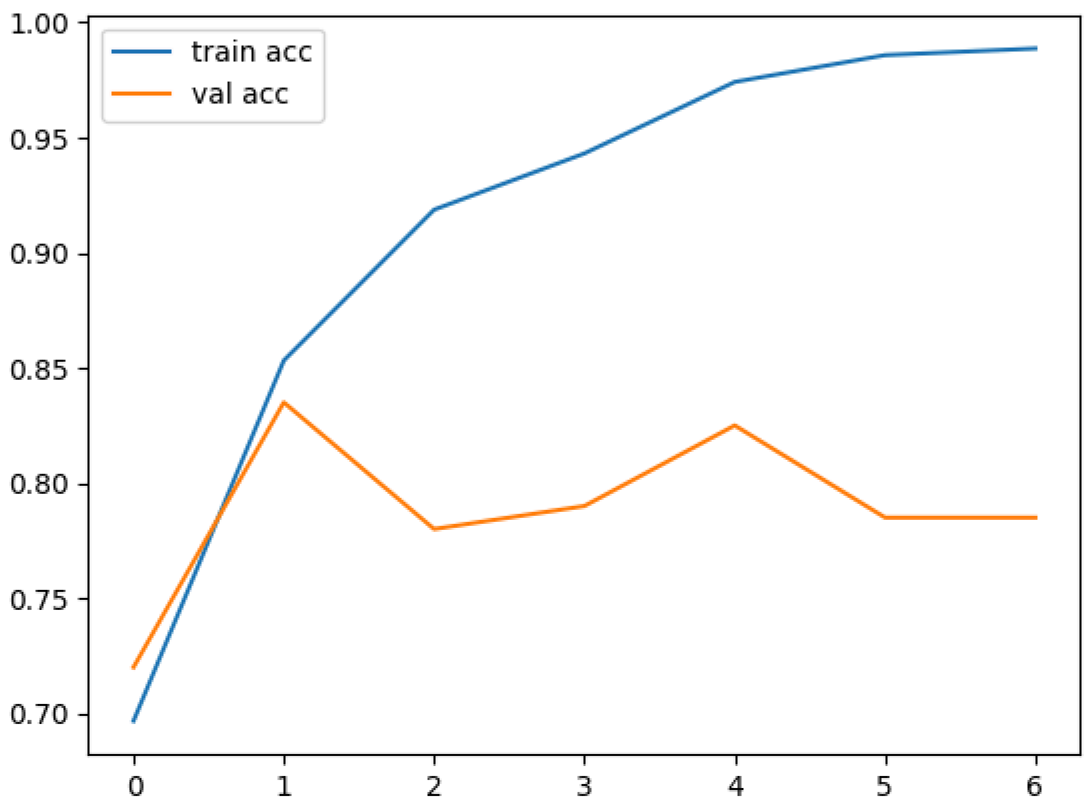
```
Epoch 1/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 403s 1s/step - accuracy: 0.6156 - loss: 1.1598 - val_ac
curacy: 0.7200 - val_loss: 0.7083
Epoch 2/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 394s 1s/step - accuracy: 0.8594 - loss: 0.3956 - val_ac
curacy: 0.8350 - val_loss: 0.5191
Epoch 3/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 392s 1s/step - accuracy: 0.9338 - loss: 0.2116 - val_ac
curacy: 0.7800 - val_loss: 0.6926
Epoch 4/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 400s 1s/step - accuracy: 0.9344 - loss: 0.1965 - val_ac
curacy: 0.7900 - val_loss: 0.6009
Epoch 5/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 401s 1s/step - accuracy: 0.9767 - loss: 0.1015 - val_ac
curacy: 0.8250 - val_loss: 0.5563
Epoch 6/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 405s 1s/step - accuracy: 0.9918 - loss: 0.0576 - val_ac
curacy: 0.7850 - val_loss: 0.6361
Epoch 7/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 406s 1s/step - accuracy: 0.9902 - loss: 0.0575 - val_ac
curacy: 0.7850 - val_loss: 0.6194
Epoch 7: early stopping
```

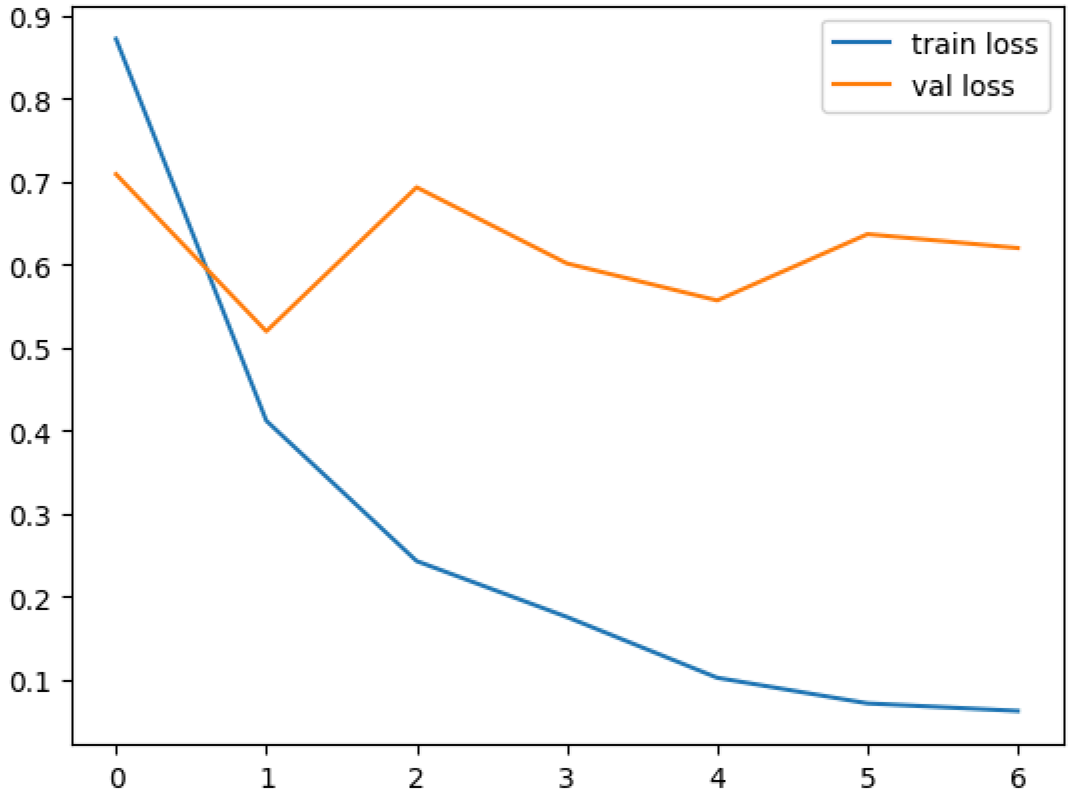
Efterbehandling

Det kan vi så plotte for at se hvordan den klarer præcision og loss.

Her kan man se at den generelt bliver bedre dog er det minimalt hvor meget man får efter de første epoker.
Man kan også se at den ikke er specielt præcis hvilket skyldes manglen på processorkraft så man kan kører i større opløsning samt manuelt datavalg.

In [ ]:
```
# accuracies
plt.plot(result.history['accuracy'], label='train acc')
plt.plot(result.history['val_accuracy'], label='val acc')
plt.legend()
# plt.savefig('vgg-acc-rps-1.png')
plt.show()
# loss
plt.plot(result.history['loss'], label='train loss')
plt.plot(result.history['val_loss'], label='val loss')
plt.legend()
# plt.savefig('vgg-loss-rps-1.png')
plt.show()
```





Vi kan til sidst gemme modelen så man kan bruge den senere og så kan man loade den igen.

In [ ]:
```
model.save("model.keras")
```

In [ ]:
```
from tensorflow.keras.models import load_model

model = load_model("model.keras")
```

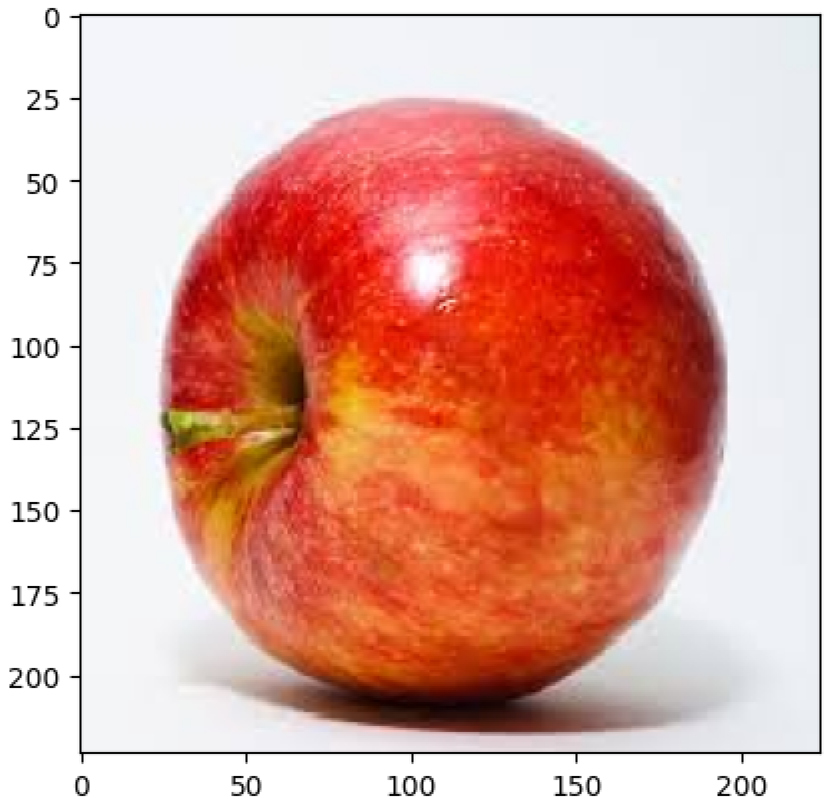
Vi kan også se data over præcision for hver enkel frugt.
Her kan man se at den generelt er præcis og man ser ikke den store svingning undtagen for bananer dog skyldes det nok at vores test data kun er på 100 billeder.

In [ ]:
```
from sklearn.metrics import classification_report,confusion_matrix
import numpy as np
#predict
y_pred=model.predict(test_x)
y_pred=np.argmax(y_pred,axis=1)

print(classification_report(y_pred,test_y))
print(confusion_matrix(y_pred,test_y))
```

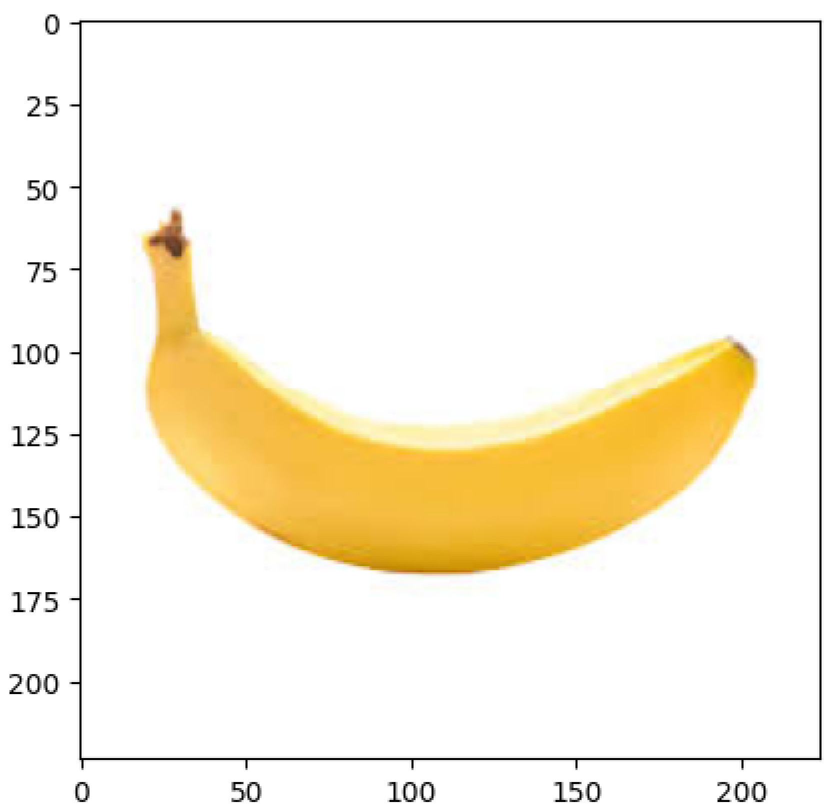
```
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 942ms/step
              precision    recall  f1-score   support

           0       0.80      0.89      0.84        18
           1       1.00      0.91      0.95        22
           2       0.90      0.72      0.80        25
           3       0.85      0.85      0.85        20
           4       0.75      1.00      0.86        15

    accuracy                           0.86       100
   macro avg       0.86      0.87      0.86       100
weighted avg       0.87      0.86      0.86       100

[[16  0  1  1  0]
 [ 0 20  1  1  0]
 [ 2  0 18  1  4]
 [ 2  0  0 17  1]
 [ 0  0  0  0 15]]
```

Her kan vi også teste med billeder der er helt nye fundet fra google.
Mønsteret fortsætter her hvor de fleste virker men at den har meget svært ved at finde ud af hvad vindruer er.
Det skyldes at i den lave opløsning kan de nemt ligner et æble eller en mango.

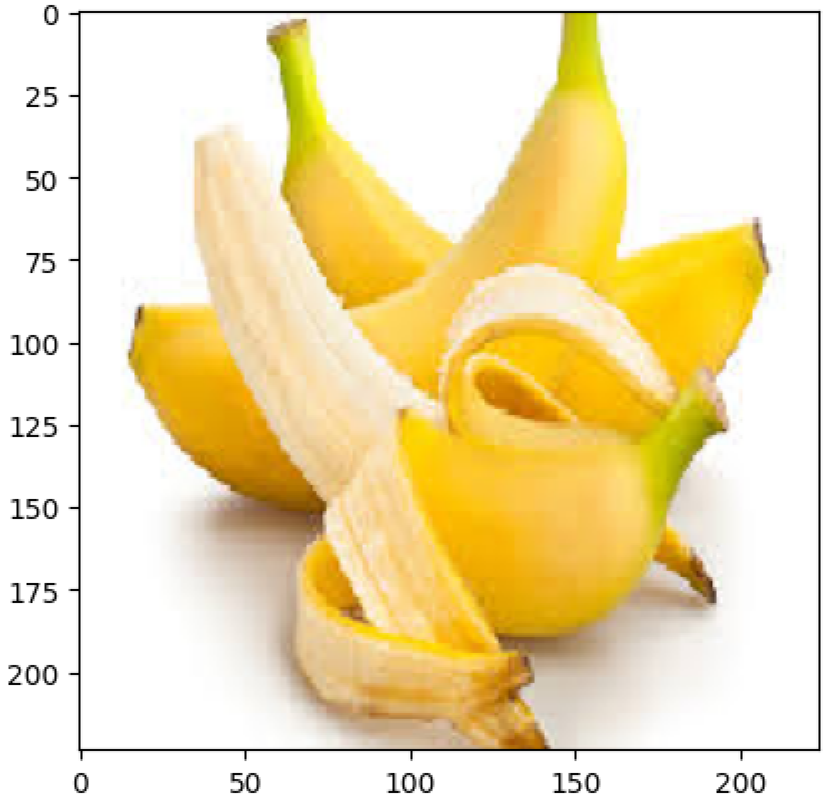
In [ ]:
```
path="test"
for img in os.listdir(path):
    img=image.load_img(path+"/"+img,target_size=(224,224))
    plt.imshow(img)
    plt.show()
    x=image.img_to_array(img)
    x=np.expand_dims(x,axis=0)
    images=np.vstack([x])
    pred=model.predict(images,batch_size=1)
    print(pred)
    if pred[0][0]>0.5:
        print("Apple")
    elif pred[0][1]>0.5:
        print("Banan")
    elif pred[0][2]>0.5:
        print("Grape")
    elif pred[0][3]>0.5:
        print('Mango')
    elif pred[0][4]>0.5:
        print('Strawberry')
    else:
        print("Unknown")
```



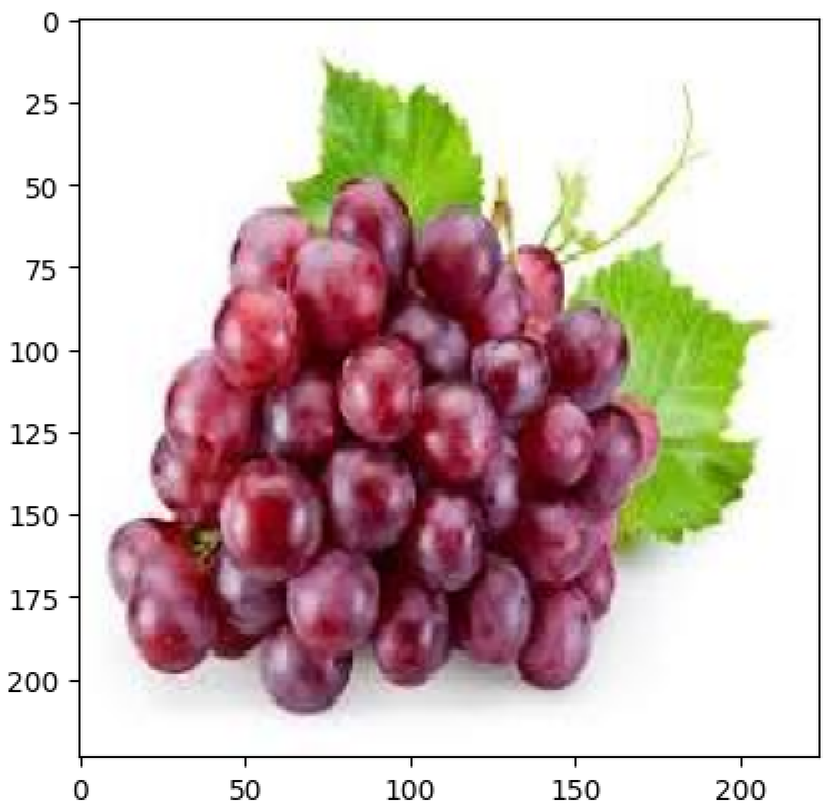

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step
[[1.0000000e+00 0.0000000e+00 5.2712115e-34 2.1184757e-33 0.0000000e+00]]
Apple
```



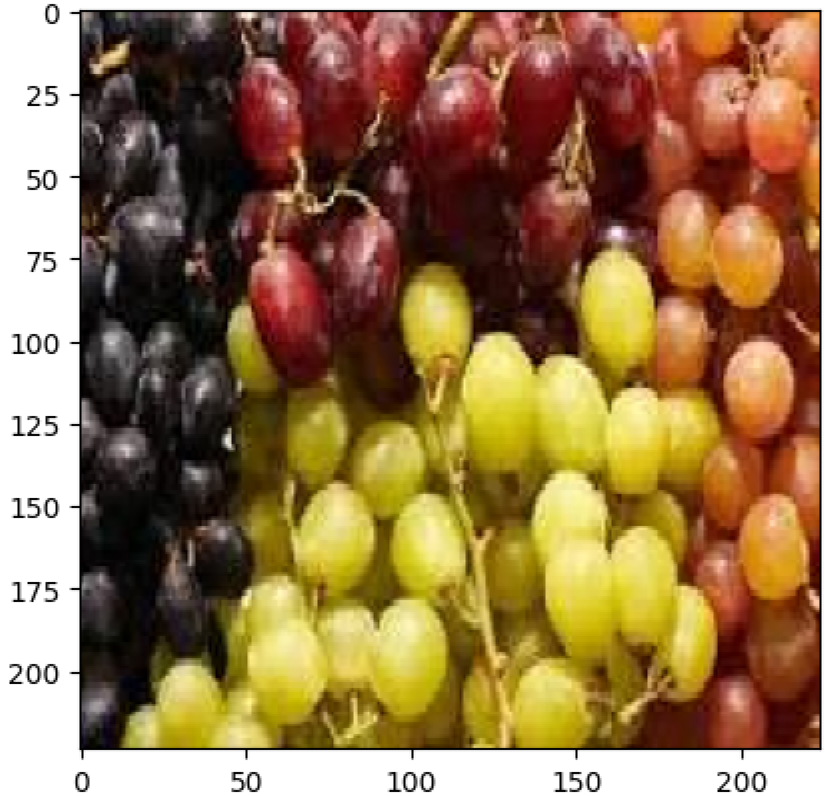

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step
[[0. 1. 0. 0. 0.]]
Banan
```



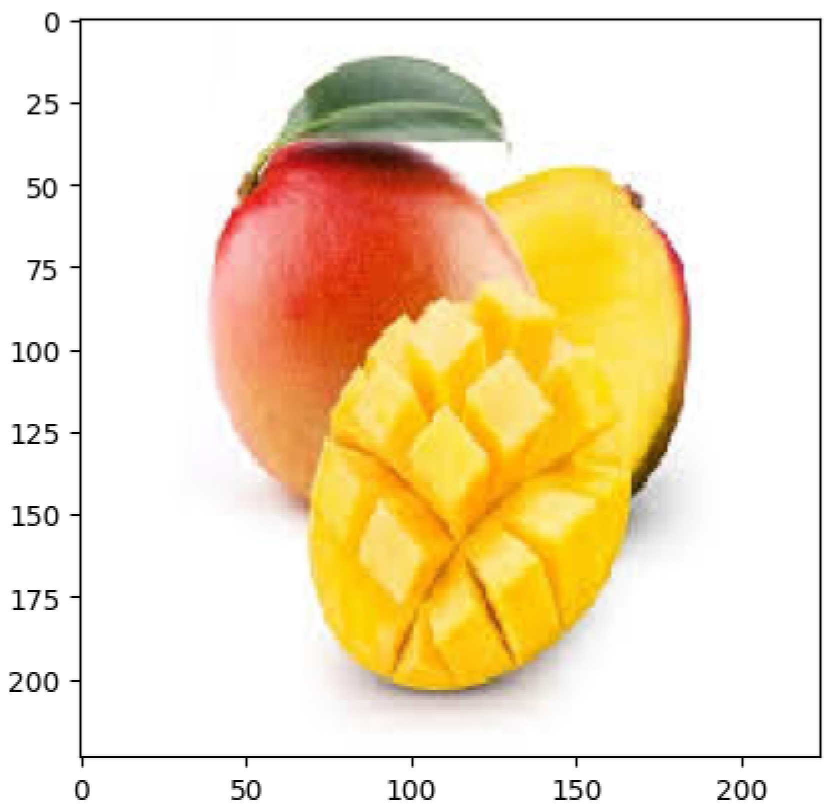

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step
[[0. 1. 0. 0. 0.]]
Banan
```



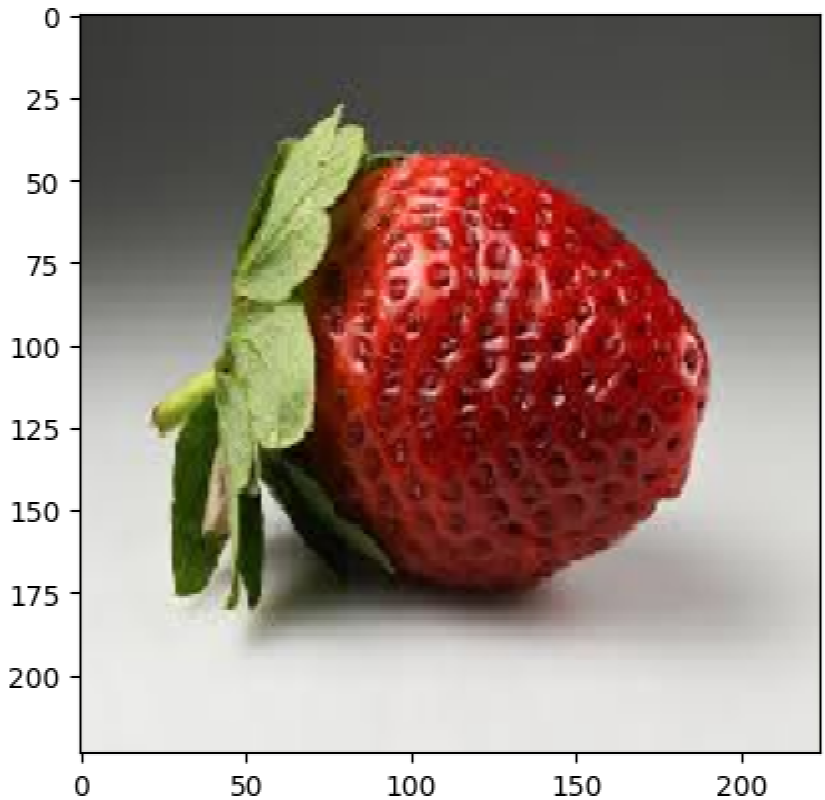

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step
[[0. 0. 1. 0. 0.]]
Grape
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step
[[2.1070646e-14 0.0000000e+00 1.8383660e-03 9.9816161e-01 0.0000000e+00]]
Mango
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
[[0. 0. 0. 1. 0.]]
Mango
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step
[[0. 0. 0. 0. 1.]]
Strawberry
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step
[[0. 0. 0. 0. 1.]]
Strawberry
```

Data augmentation

Vi havde også fået til opgave at implementerer data augmentation men da vi både havde problemer med at køre programmet med så mange ekstra billeder og da det viser sig at `Keras` som standard implementerer data augmentation valgte vi ikke at gøre det selv. Derfor har vi ikke implementeret det men valgt stadig at vise hvordan det kan fungerer.

In [ ]:
```
image, label = next(iter(training_set))
print(np.max(image[0]))
plt.imshow(image[0])
```

```
1.0
```

Out[ ]: `<matplotlib.image.AxesImage at 0x26ecfe5bd10>`

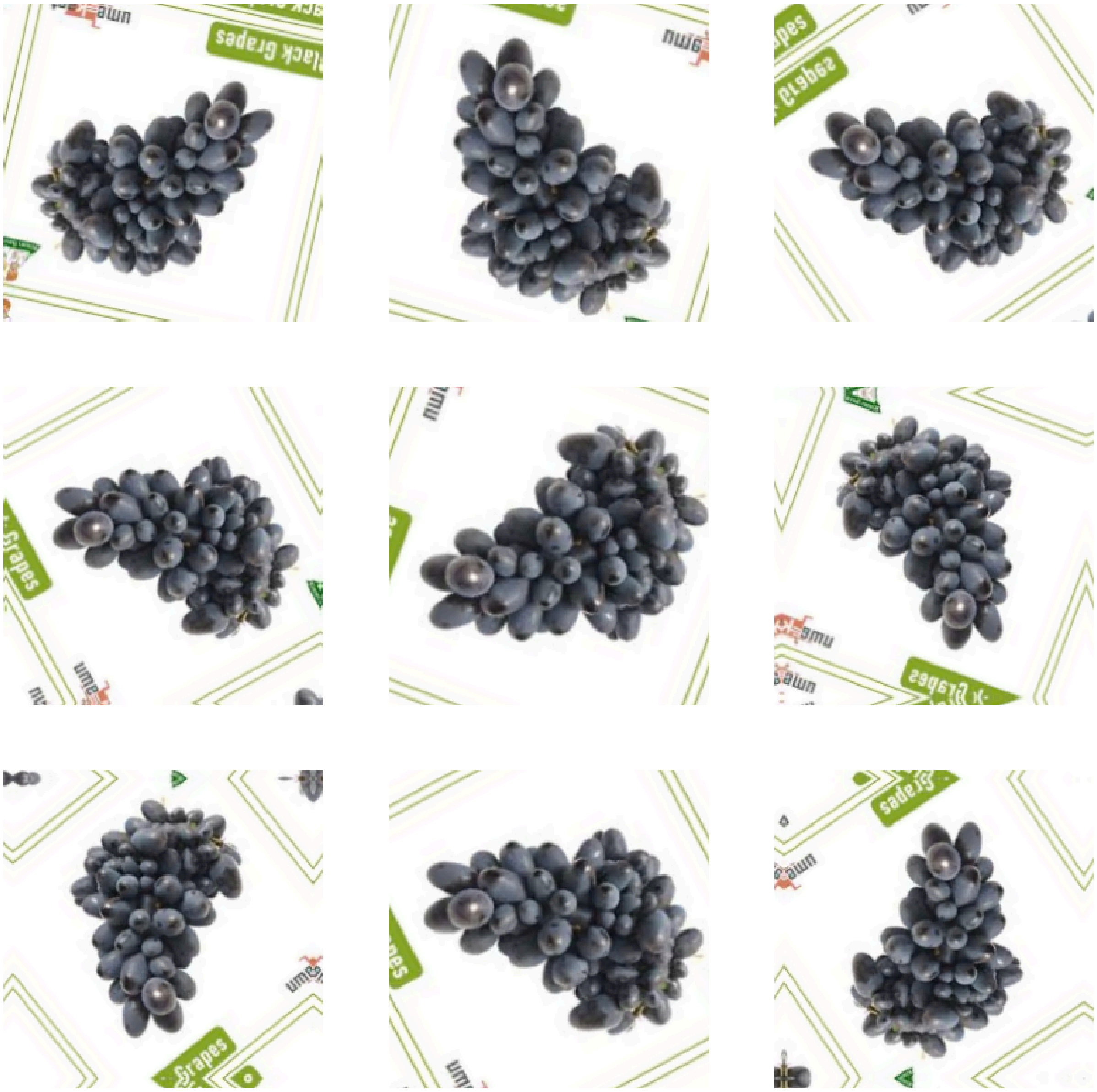
Med billedet kan vi fokuserer og ændrer opløsningen så de har den samme og bevarer fokuset på objektet.

In [ ]:
```
resize_and_rescale = Sequential([
  Resizing(224, 224),
])

result = resize_and_rescale(image[0])
print(result.shape)

plt.imshow(result)

print("Min and max pixel values:", result.numpy().min(), result.numpy().max())
```

```
(224, 224, 3)
Min and max pixel values: 0.0 1.0
```

Her kan vi så lave data augmentation der går ud på at ændrer hvordan billederne er orienteret, zoomet, osv.

In [ ]:
```
data_augmentation = Sequential([
  RandomFlip("horizontal_and_vertical"),
  RandomRotation(0.2),
  RandomZoom(0.2),
])

image = cast(expand_dims(result, 0), float32)

plt.figure(figsize=(10, 10))
for i in range(9):
  augmented_image = data_augmentation(image)
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(augmented_image[0])
  plt.axis("off")
```



Et andet problem ved denne metode er at der bruger vi en funktion til at gøre alle billederne klar. Men den funktion kræver at man bruger Tensorflows egen dataset struktur som er en meget specifik opsætning som vores dataset ikke kom med.
Derfor ville det også kræve at vi laver om på vores dataset.

Funktionen står for at resize alle billeder ligesom vi selv manuelt har gjordt længere oppe samt køre den data_augmentationsfunktion som vi skrev før på træningssættet.

In [ ]:
```
batch_size = 32
AUTOTUNE = tf.data.AUTOTUNE

def prepare(ds, shuffle=False, augment=False):
  # Resize and rescale all datasets.
  # ds = ds.map(lambda x, y: (resize_and_rescale(x), y),
  #             num_parallel_calls=AUTOTUNE)

  if shuffle:
```

```
    ds = ds.shuffle(1000)

  # Batch all datasets.
  ds = ds.batch(batch_size)

  # Use data augmentation only on the training set.
  if augment:
    ds = ds.map(lambda x, y: (data_augmentation(x, training=True), y),
                num_parallel_calls=AUTOTUNE)

  # Use buffered prefetching on all datasets.
  return ds.prefetch(buffer_size=AUTOTUNE)
```

Vi valgte at fikse problemet med datastrukturen ved at lave den om til et Tensorflow dataset ved hjælp af funktionen `from_tensor_slices` der tager vores x og y værdier og laver et samlet `DatasetV2` der er tensorflows egen datastruktur.

In [ ]:
```
train_x = Dataset.from_tensor_slices((train_x, train_y))
test_x = Dataset.from_tensor_slices((test_x, test_y))
val_x = Dataset.from_tensor_slices((val_x, val_y))
```

Det kan vi så bruge i prepare så vi får vores augmenterede data ud

In [ ]:
```
out_train = prepare(train_x, shuffle=True, augment=True)
out_test = prepare(test_x)
out_val = prepare(val_x)
```

Her kan man også se hvordan den har en ekstra dimension der skyldes de augmenterede data.

In [ ]:
```
print(out_train.as_numpy_iterator().next()[0].shape, train_x.as_numpy_iterator().ne
```

```
(32, 224, 224, 3) (224, 224, 3) [array([[[[0.33598536, 0.2178674 , 0.22063573],
         [0.34309334, 0.22049826, 0.22473282],
         [0.345332  , 0.22257896, 0.22725658],
         ...,
         [0.4335565 , 0.26387337, 0.23899895],
         [0.42694622, 0.25879166, 0.23270884],
         [0.42032304, 0.2566176 , 0.23010482]],

        [[0.35282165, 0.23526412, 0.23763227],
         [0.35426268, 0.23062484, 0.23454641],
         [0.35546532, 0.23183724, 0.23575881],
         ...,
         [0.44299936, 0.2697915 , 0.24205047],
         [0.4354233 , 0.26313108, 0.2344429 ],
         [0.43042985, 0.2578637 , 0.22990133]],

        [[0.35358167, 0.23995647, 0.241539  ],
         [0.35770762, 0.23835307, 0.24186356],
         [0.36429727, 0.24101633, 0.24473277],
         ...,
         [0.45495045, 0.27710038, 0.24676524],
         [0.44613206, 0.27052596, 0.23771504],
         [0.43868604, 0.2604108 , 0.227568  ]],

        ...,

        [[0.11924341, 0.09387992, 0.1820627 ],
         [0.11542722, 0.09291613, 0.17888504],
         [0.10360736, 0.08345408, 0.16642861],
         ...,
         [0.23446068, 0.15995088, 0.14034304],
         [0.25040838, 0.17589855, 0.15629071],
         [0.27006245, 0.19555265, 0.1759448 ]],

        [[0.1188418 , 0.08855607, 0.17855093],
         [0.12098016, 0.09518035, 0.18473032],
         [0.11700419, 0.09450699, 0.18204525],
         ...,
         [0.22456364, 0.15005384, 0.13044599],
         [0.24575543, 0.17124563, 0.15163781],
         [0.27274302, 0.19823322, 0.17862538]],

        [[0.12420332, 0.0852427 , 0.17652223],
         [0.12156717, 0.09055914, 0.18075511],
         [0.12147276, 0.09601671, 0.18530306],
         ...,
         [0.2152739 , 0.13991678, 0.12030894],
         [0.24821189, 0.1733891 , 0.15378127],
         [0.27155134, 0.19704154, 0.1774337 ]]],


       [[[0.7137255 , 0.6901961 , 0.96862745],
         [0.71372557, 0.6901961 , 0.96862745],
         [0.71372557, 0.6901961 , 0.96862745],
         ...,
         [1.        , 0.9960824 , 1.        ],
```

```
         [1.        , 0.9960882 , 1.        ],
         [1.        , 0.9960842 , 1.        ]],

        [[0.71372557, 0.69019616, 0.96862745],
         [0.71372557, 0.6901961 , 0.96862745],
         [0.71372557, 0.6901961 , 0.96862745],
         ...,
         [1.        , 0.99645245, 1.        ],
         [1.        , 0.9970104 , 1.        ],
         [1.        , 0.9966327 , 1.        ]],

        [[0.7155018 , 0.6919724 , 0.9704038 ],
         [0.7149417 , 0.6914122 , 0.96984357],
         [0.71438146, 0.690852  , 0.9692834 ],
         ...,
         [1.        , 0.99840426, 0.9983714 ],
         [1.        , 0.9985571 , 0.99881464],
         [1.        , 0.99747896, 0.99981636]],

        ...,

        [[0.8117647 , 0.7882353 , 0.9960784 ],
         [0.8117647 , 0.7882353 , 0.99607843],
         [0.8117647 , 0.7882353 , 0.9960784 ],
         ...,
         [0.8214513 , 0.79792184, 0.9822355 ],
         [0.81960785, 0.79607844, 0.9803921 ],
         [0.81960785, 0.79607844, 0.9803922 ]],

        [[0.8117647 , 0.7882353 , 0.9960785 ],
         [0.8117647 , 0.7882353 , 0.9960784 ],
         [0.8117647 , 0.7882353 , 0.9960784 ],
         ...,
         [0.82089114, 0.7973617 , 0.9816754 ],
         [0.81960785, 0.79607844, 0.9803921 ],
         [0.81960785, 0.79607844, 0.98039216]],

        [[0.8117647 , 0.7882353 , 0.99607843],
         [0.8117647 , 0.7882353 , 0.99607843],
         [0.8117647 , 0.7882353 , 0.9960785 ],
         ...,
         [0.8203308 , 0.7968014 , 0.9811152 ],
         [0.81960785, 0.79607844, 0.98039216],
         [0.81960785, 0.79607844, 0.98039216]]],


       [[[0.3699963 , 0.30371392, 0.13874319],
         [0.37032703, 0.2963283 , 0.13578174],
         [0.3705536 , 0.29482523, 0.13535477],
         ...,
         [0.20147522, 0.19056943, 0.2342084 ],
         [0.19879678, 0.19072266, 0.23229139],
         [0.18657337, 0.19245747, 0.22183697]],

        [[0.3693983 , 0.30635744, 0.14046754],
         [0.36747563, 0.29829627, 0.13825808],
```

```
  [0.3674711 , 0.2967265 , 0.13813561],
  ...,
  [0.20682111, 0.1855984 , 0.24908325],
  [0.20415567, 0.18609902, 0.24610117],
  [0.19265264, 0.18821147, 0.2308125 ]],

 [[0.369483  , 0.30706605, 0.14079222],
  [0.36702257, 0.2990237 , 0.13904043],
  [0.36694705, 0.2974785 , 0.13905752],
  ...,
  [0.20820805, 0.18469742, 0.25185126],
  [0.20551078, 0.18525377, 0.24863341],
  [0.19403848, 0.18737327, 0.23224391]],

 ...,

 [[0.2784314 , 0.3647059 , 0.5019608 ],
  [0.27580515, 0.36207962, 0.49933454],
  [0.2750705 , 0.361345  , 0.4985999 ],
  ...,
  [0.17762165, 0.31068712, 0.552753  ],
  [0.17938226, 0.314561  , 0.560922  ],
  [0.18751927, 0.33078122, 0.5943146 ]],

 [[0.27763712, 0.36391163, 0.5011666 ],
  [0.27485433, 0.361086  , 0.49834093],
  [0.27414557, 0.36036524, 0.49762017],
  ...,
  [0.17963937, 0.31649694, 0.5632754 ],
  [0.18126878, 0.31984147, 0.57017845],
  [0.18867873, 0.3338489 , 0.59810066]],

 [[0.27417246, 0.36016202, 0.4974169 ],
  [0.27154553, 0.3573464 , 0.49457926],
  [0.27105471, 0.3568278 , 0.49405456],
  ...,
  [0.1879507 , 0.33896786, 0.6033193 ],
  [0.18883593, 0.3400156 , 0.60465336],
  [0.1921185 , 0.34408626, 0.6078421 ]]],


...,


[[[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ]],

 [[1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  [1.        , 1.        , 1.        ],
  ...,
```

```
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ]],

         [[1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          ...,
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ]],

         ...,

         [[1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          ...,
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ]],

         [[1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          ...,
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ]],

         [[1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          ...,
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ]]],


        [[[1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          ...,
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ]],

         [[1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          ...,
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ],
          [1.        , 1.        , 1.        ]],

         [[1.        , 1.        , 1.        ],
```

```
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        ...,
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ],
        [1.        , 1.        , 1.        ]],

       ...,

       [[0.98615676, 0.99981266, 0.97754604],
        [0.961317  , 0.93558824, 0.84262407],
        [0.8640425 , 0.7602279 , 0.5616577 ],
        ...,
        [0.9996897 , 1.        , 1.        ],
        [1.        , 0.99991614, 1.        ],
        [1.        , 0.9994584 , 1.        ]],

       [[0.9825084 , 0.9995054 , 0.99337643],
        [0.9862858 , 0.9879904 , 0.9341974 ],
        [0.9120937 , 0.8388592 , 0.68158364],
        ...,
        [0.9981089 , 1.        , 0.9999973 ],
        [0.99985445, 0.9996534 , 1.        ],
        [1.        , 0.9982259 , 1.        ]],

       [[0.9819187 , 0.99853855, 0.9999934 ],
        [0.9893127 , 0.9997846 , 0.968938  ],
        [0.9415581 , 0.89880407, 0.78424555],
        ...,
        [0.9937188 , 1.        , 0.99990034],
        [0.9989843 , 0.9993227 , 1.        ],
        [0.9998704 , 0.99678254, 1.        ]]],


      [[[0.11443734, 0.15837027, 0.38250506],
        [0.13490248, 0.18109621, 0.4004855 ],
        [0.1806552 , 0.22852527, 0.44371802],
        ...,
        [0.2862503 , 0.36729568, 0.3985681 ],
        [0.28295907, 0.36492166, 0.39626357],
        [0.280263  , 0.36265254, 0.39398426]],

       [[0.10892002, 0.15416338, 0.37555602],
        [0.12885381, 0.17766735, 0.39478603],
        [0.17022616, 0.2215536 , 0.43395507],
        ...,
        [0.28256392, 0.36502746, 0.39541587],
        [0.2793017 , 0.36175296, 0.39277896],
        [0.27564114, 0.35871857, 0.3893485 ]],

       [[0.10152636, 0.14979781, 0.3680347 ],
        [0.12181996, 0.17263302, 0.38727406],
        [0.15509266, 0.20882124, 0.41836876],
        ...,
        [0.28016508, 0.36321533, 0.3913227 ],
        [0.27602953, 0.3591048 , 0.38892803],
```

```
          [0.27178857, 0.35643584, 0.38500112]],

         ...,

         [[0.60225403, 0.5476125 , 0.58558196],
          [0.60664934, 0.5532905 , 0.5890762 ],
          [0.6082189 , 0.5567442 , 0.59155786],
          ...,
          [0.52789456, 0.5166515 , 0.5530998 ],
          [0.5438931 , 0.53288484, 0.56693655],
          [0.5574487 , 0.5473027 , 0.5783965 ]],

         [[0.58560586, 0.53223884, 0.5698756 ],
          [0.5850766 , 0.5325943 , 0.5682752 ],
          [0.5830607 , 0.5320224 , 0.56502485],
          ...,
          [0.48638147, 0.47785488, 0.51648676],
          [0.5112403 , 0.5034374 , 0.53883666],
          [0.53285086, 0.52474666, 0.5569569 ]],

         [[0.55882704, 0.5072478 , 0.5429739 ],
          [0.55780804, 0.5064693 , 0.5407087 ],
          [0.55756444, 0.50658405, 0.53830385],
          ...,
          [0.4385989 , 0.43185654, 0.47355425],
          [0.47075725, 0.4645561 , 0.5025256 ],
          [0.5003404 , 0.49304688, 0.5281617 ]]]], dtype=float32), array([0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0])] [array([[[[1.        , 0.98431373, 0.9803921
6],
         [1.        , 0.98431373, 0.98039216],
         [1.        , 0.98431373, 0.98039216],
         ...,
         [1.        , 0.99607843, 0.99215686],
         [1.        , 0.99607843, 0.99215686],
         [1.        , 0.99607843, 0.99215686]],

        [[1.        , 0.98431373, 0.98039216],
         [1.        , 0.98431373, 0.98039216],
         [1.        , 0.98431373, 0.98039216],
         ...,
         [1.        , 0.99607843, 0.99215686],
         [1.        , 0.99607843, 0.99215686],
         [1.        , 0.99607843, 0.99215686]],

        [[1.        , 0.98431373, 0.98039216],
         [1.        , 0.98431373, 0.98039216],
         [1.        , 0.98431373, 0.98039216],
         ...,
         [1.        , 0.99607843, 0.99215686],
         [1.        , 0.99607843, 0.99215686],
         [1.        , 0.99607843, 0.99215686]],

        ...,

        [[0.98823529, 0.96862745, 0.96470588],
```

```
  [0.98823529, 0.96862745, 0.96470588],
  [0.98823529, 0.96862745, 0.96470588],
  ...,
  [0.99607843, 0.98823529, 0.98431373],
  [0.99607843, 0.98823529, 0.98431373],
  [0.99607843, 0.98823529, 0.98431373]],

 [[0.98823529, 0.96862745, 0.96470588],
  [0.98823529, 0.96862745, 0.96470588],
  [0.98823529, 0.96862745, 0.96470588],
  ...,
  [0.99607843, 0.98823529, 0.98431373],
  [0.99607843, 0.98823529, 0.98431373],
  [0.99607843, 0.98823529, 0.98431373]],

 [[0.98823529, 0.96862745, 0.96470588],
  [0.98823529, 0.96862745, 0.96470588],
  [0.98823529, 0.96862745, 0.96470588],
  ...,
  [0.99607843, 0.98823529, 0.98431373],
  [0.99607843, 0.98823529, 0.98431373],
  [0.99607843, 0.98823529, 0.98431373]]]), 0]
```

Vi valgte dog ikke at gå med den da vi kunne se at den klarede sig meget dårligerer end vores normale model og at det nærmest virker som om den gætter ud fra loss værdien.

In [ ]:
```
result = model.fit(
  out_train,
  validation_data=out_val,
  epochs=10,
  callbacks=[early_stop],
  batch_size=32,shuffle=True)
```

```
Epoch 1/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 662s 2s/step - accuracy: 0.8195 - loss: 1.1504 - val_ac
curacy: 0.4750 - val_loss: 2.7804
Epoch 2/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 791s 3s/step - accuracy: 0.8289 - loss: 1.6257 - val_ac
curacy: 0.4950 - val_loss: 3.9848
Epoch 3/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 697s 2s/step - accuracy: 0.8346 - loss: 1.8824 - val_ac
curacy: 0.5400 - val_loss: 5.6249
Epoch 4/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 1048s 3s/step - accuracy: 0.8204 - loss: 3.1259 - val_a
ccuracy: 0.5400 - val_loss: 5.0240
Epoch 5/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 839s 3s/step - accuracy: 0.8345 - loss: 2.2281 - val_ac
curacy: 0.6250 - val_loss: 6.5897
Epoch 6/10
304/304 ━━━━━━━━━━━━━━━━━━━━ 1198s 4s/step - accuracy: 0.8145 - loss: 3.7871 - val_a
ccuracy: 0.6150 - val_loss: 4.8353
Epoch 6: early stopping
```