Ivan Catalano

Bruno Silva

Tillämpad AI, datautvinning, maskininlärning och deep learning

AI Developer HT 2022

JENSEN Yrkeshögskolan

Rapport

Deep Learning Image Classification Projekt

Sammanfattning

Detta projekt syftar till att demonstrera användningen av djupinlärning (deep learning) inom bildklassificering med hjälp av Kaggle-datasetet "Pistagenötter". I detta projekt fokuserar vi på att utveckla en djupinlärningsmodell som kan klassificera olika variationer av pistagekärnor baserat på bilder.

Bildklassificering med djupinlärning är en viktig tillämpning av artificiell intelligens. Denna teknik har många användningsområden, inklusive medicinsk bildigenkänning, autonoma fordon och kvalitetskontroll av livsmedel. I detta projekt kommer vi att fokusera på kvalitetskontroll av livsmedel och klassificera pistagekärnor i olika kategorier baserat på deras visuella egenskaper.

Innehållsförteckning

Sammanfattning 2

Innehållsförteckning 3

Teori 4

Datahantering och förberedelse .....4 - 7

Logistisk regression 7

Djup inlärning med Neural Networks .....7 – 9

Data preprocessing .....9 - 10

Djup inlärning med Convolutional Neural Network (CNN) .....11 - 12

Djup inlärning med Transfer Learning (VGG16 och MobileNetV2) .....13 - 17

Resultat och Utvärdering .....18 - 20

Slutsatser 20

Teori

Datahantering och förberedelse

Vi importerade biblioteken som var nödiga till detta projekt

```
# bibliotek för att ignorera varningar
import warnings
warnings.filterwarnings('ignore')

# importering av bibliotek som behövsimport matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import sys

# bibliotek från funkiotner.py
import funktioner as fn

# bibliotek för Deep Learning
import tensorflow as tf

# bibliotek för machine learning
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, log_loss
```

Pistacium datasetet är ett dataset som innehåller bilder om pistacium nötter. Detta dataset innehåller också en excel-fil med en lista över pistaschmätningar indelade i två klasser.

Vad vi gjorde:
- Vi har använt ett dataset från Kaggle som heter "muratkokludataset/pistachio-image-dataset"
- Vi har kört en lite EDA på datasetet av attributerna för att se hur det ser ut.
- Vi har prövat dela upp datasetet i clustrar och se hur det ser ut, med en PCA analys också.
- Vi har kört en Machine Learning-modell på dataset och jämförde det med en Deep Learning-modell.

Först laddade vi in och utforskade datasetet som var i form av en Excel-fil. Vi inspekterade datasetets storlek och dess kolumnstruktur.

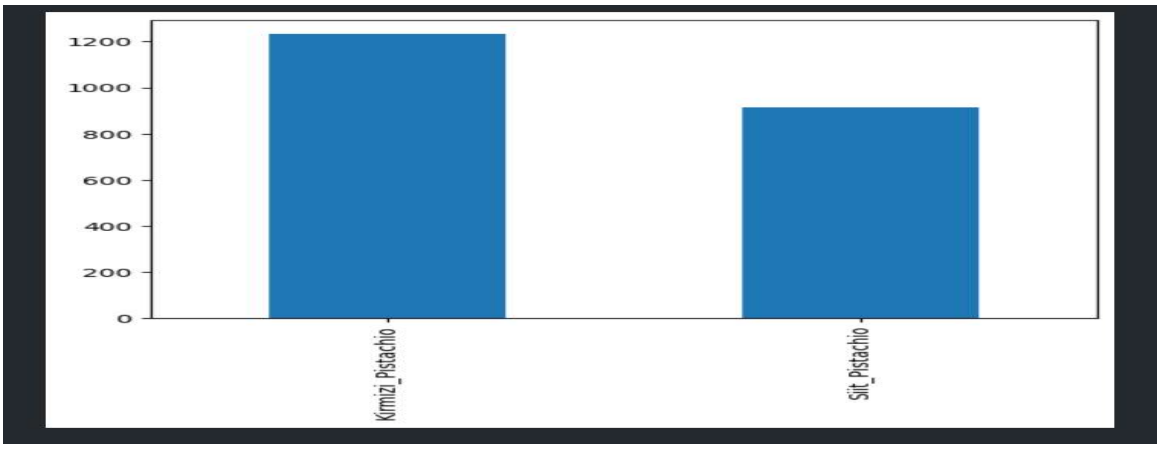
```
# läsa in data från csv-fil och skapa en dataframe
df = pd.read_excel(PATH +'/dataset/dataset.xlsx', sheet_name='Pistachio_Dataset')


# printa ut första 5 raderna i datasetena
df.head()
```

	AREA	PERIMETER	MAJOR_AXIS	MINOR_AXIS	ECCENTRICITY	EQDIASQ	SOLIDITY	CONVEX_AREA
0	63391	1568.405	390.3396	236.7461	0.7951	284.0984	0.8665	73160
1	68358	1942.187	410.8594	234.7525	0.8207	295.0188	0.8765	77991
2	73589	1246.538	452.3630	220.5547	0.8731	306.0987	0.9172	80234
3	71106	1445.261	429.5291	216.0765	0.8643	300.8903	0.9589	74153
4	80087	1251.524	469.3783	220.9344	0.8823	319.3273	0.9657	82929

Datasetet innehöll information om olika kategorier av pistachionötter, och vår uppgift var att klassificera dem baserat på bilderna.

Vi använde bibliotek som Pandas och Matplotlib för att utföra dataanalys och visualisering.

```
# printa ut grafiken som visar hur många rader det finns av varje klass
df['Class'].value_counts().plot(kind='bar')
```

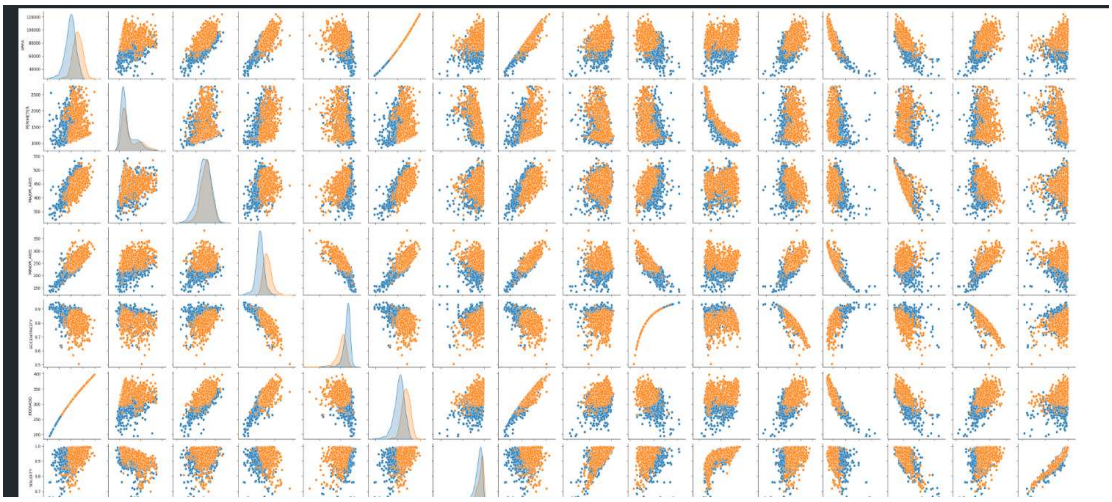


För att upptäkta om det finns någon korrelation mellan kolumner i datasetet har vi använt oss av en pairplot matris.

Vi har också använt oss av en PCA-analys för att se om det är möjligt att dela upp datasetet i clustrar.

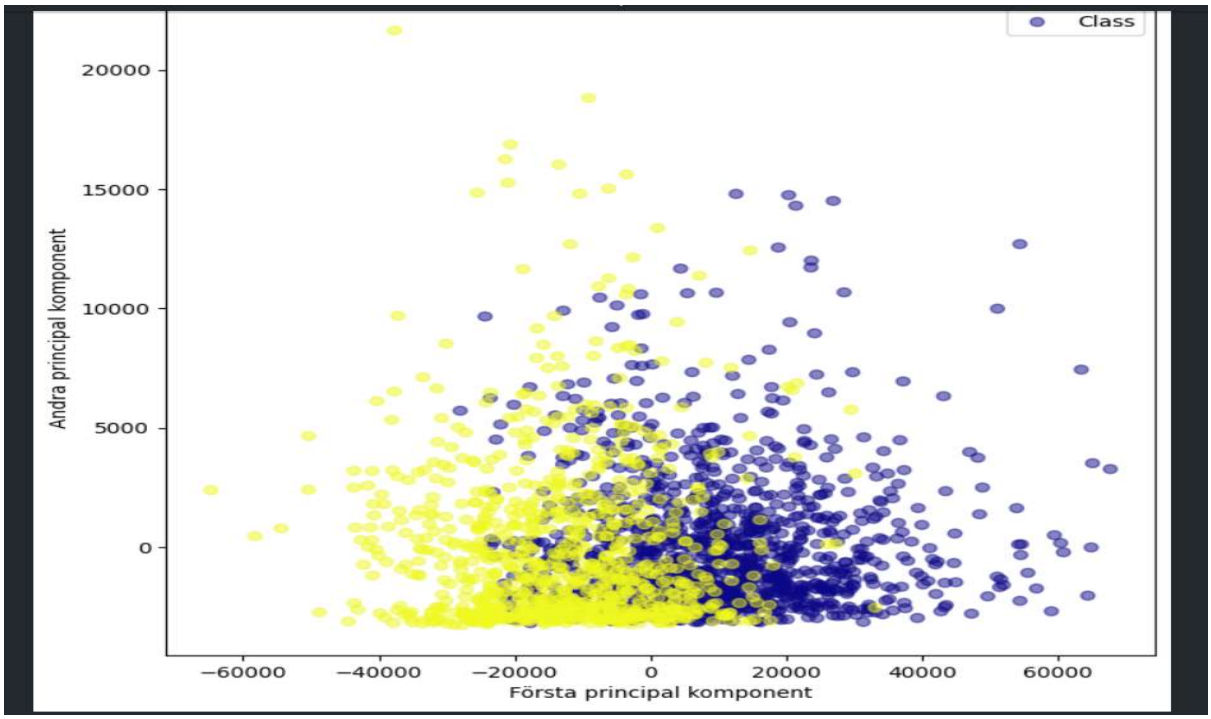
Vi skapade en stapeldiagram för att visa antalet bilder per klass och använde även en pairplot-matris och PCA för att upptäcka möjlig korrelation och gruppering i datasetet.

```
# Pairplot
df_cluster = df.copy()
cluster_classes = df_cluster['Class'].unique()
df_cluster['Class'].replace(cluster_classes,[0,1])

sns.pairplot(df_cluster, hue='Class')
plt.show()
```



```
# PCA
# spara klasserna i en variabel
classes = df['Class'].unique()

# skapa en instans av PCA
pca = PCA(n_components=2)

# skapa en variabel för X och y
X = df.drop('Class', axis=1)
colors = df['Class'].replace(classes,[0,1])

# träna PCA
pca.fit(X)

# transformera data
X_pca = pca.transform(X)

# kolla shape och printa ut information
print(f'Vi har {len(classes)} klasser: {classes}, och {len(df.columns)} kolumner. Med PCA kan vi minska antalet dimensioner till {pca.n_components}.')
print('\n')
print("original shape:   ", X.shape)
print("transformed shape:", X_pca.shape)

# plotta PCA
plt.figure(figsize=(8,8))
plt.scatter(x=X_pca[:,0], y=X_pca[:,1], c=colors, cmap='plasma', alpha=0.5, label='Class')
plt.xlabel('Första principal komponent')
plt.ylabel('Andra principal komponent')
plt.legend()
plt.show()
```

Vi har 2 klasser: ['Kirmizi_Pistachio' 'Siit_Pistachio'], och 17 kolumner. Med PCA kan vi minska antalet dimensioner till 2.



Sammanfattning av analysen:

- Pairplot har visat att det inte finns någon tydlig skillnad mellan de olika klasserna.

- PCA-analys framkom det att pistagenötter inte kunna delas upp exakt i två distinkta kluster.

Så vi har kommit fram till att det är svårt att klassificera pistagenötter med hjälp av Machine Learning-modeller, men vi prövade ändå att köra en Machine Learning-modell på datasetet. Vi har också kört en Deep Learning-modell på datasetet och jämförde resultatet med Machine Learning-modellen.

Logistisk regression

För att få en baslinje för prestanda använde vi logistisk regression för att klassificera bilderna. Vi utvärderade modellen med hjälp av log loss och accuracy. Denna modell användes som jämförelsepunkt för våra djupinlärningsmodeller.

ML - Modell: Logistic Regression:

Vi har använt en Logistic Regression-modell för att se om vi kan få en bättre klassificering av datasetet.

Sen har vi sparat Loss och Accuracy för att jämföra med en Deep Learning-modell.

```
# Skapa variabler för att lagra data
X = df.drop('Class', axis=1)
y = df['Class'].replace(classes,[0,1])


# vi provar att använda en övervakat lärande metod för att träna en modell som kan klassificera
# pistagenötter. Vi använder en Logistik Regression.
logisticRegr = LogisticRegression()
logisticRegr.fit(X, y)
predictions = logisticRegr.predict(X)

# Beräkna noggrannheten och Loss
loss_reg = log_loss(y, logisticRegr.predict_proba(X))
accuracy_reg = logisticRegr.score(X, y)
```

Djup inlärning med Neural Networks

Vi implementerade en neural network-modell med Tensorflow och Keras.

DL - Modell: Neural Network

Vi har kört en loop för att hitta den bästa modellen för detta datasetet som har Loss och Accuracy minre än Logistic Regression.

Sen har vi sparat den bästa modellen i en fil.

```
# skapa och spara modellen
if os.path.exists(PATH + '/model.h5'):
    model = tf.keras.models.load_model(PATH + '/model.h5')
    score = model.evaluate(X, y)
    loss_dl = score[0]
    accuracy_dl = score[1]
    print('Modellet finns redan.. laddar upp det')
    print('\n')
```

Modellen bestod av flera lager inklusive input-, hidden- och output-lager.

```
else:
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Dense(8, input_dim=X.shape[1], activation='relu', name='input'))
    model.add(tf.keras.layers.Dense(12, activation='relu', name='hidden1'))
    model.add(tf.keras.layers.Dense(12, activation='relu', name='hidden2'))
    model.add(tf.keras.layers.Dense(1, activation='sigmoid', name='output'))
```

Vi använde en binär korsentropi som förlustfunktion och Adam-optimerare för träning. Modellen tränades iterativt och övervakades för att uppnå en loss som var mindre än den som uppnåddes med logistisk regression och en accuracy som var större än den som uppnåddes med logistisk regression.

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print('Modell skapas... ')
print('Training...')
print(f"vi måste få en loss mindre än {loss_reg} och en accuracy större än {accuracy_reg}")
print('...')
print('\n')
```

Träning körs med 1000 epochs och batch_size av 120. Verbose var sätt till 0 för att utföras tyst utan någon utskrift.

```
history = model.fit(X, y, epochs=1000, batch_size=120, verbose=0)
```

När vi jämför resultatet från Logistic Regression och Neural Network, får vi påstår att Neural Network har en lite bättre resultat.

```
...   ================================================
      Logistic Regression
      ================================================
      Loss:  0.3230720151630863
      Accuracy:  0.86731843575419
      ------------------------------------------------
      classification report
                    precision    recall  f1-score   support

                 0       0.87      0.90      0.89      1232
                 1       0.86      0.82      0.84       916

          accuracy                           0.87      2148
         macro avg       0.87      0.86      0.86      2148
      weighted avg       0.87      0.87      0.87      2148

      ------------------------------------------------
      confusion matrix
      [[1108  124]
       [ 161  755]]
```

```
==================================================
Deep Learning
==================================================
loss: 0.29717981815338135
accuracy: 0.873836100101471
--------------------------------------------------
classification report
              precision    recall  f1-score   support

           0       0.89      0.89      0.89      1232
           1       0.85      0.85      0.85       916

    accuracy                           0.87      2148
   macro avg       0.87      0.87      0.87      2148
weighted avg       0.87      0.87      0.87      2148

--------------------------------------------------
confusion matrix
[[1098  134]
 [ 137  779]]
```

Data preprocessing

För att förbereda våra data för modellträning utförde vi nödvändiga dataförbehandlingar. Vi Skaffade en tränings och valideringsvariabel när vi delade upp våra bilder samt en batch-storlek och bildstorlek variabel som skulle användas under träning. Batch-storleken var satt till 32, och bildstorleken var 160x160 pixlar med 3 färgkanaler (RGB). Sedan skaffade vi två variabler för att laga tränings och validations data med tensorflow keras för att förbättra träningsprocessen.

- Utför nödvändiga dataförbehandlingssteg på bilddatauppsättningen.
- Tillämpa tekniker som storleksändring, normalisering och förstärkning för att förbättra utbildningsdatas kvalitet och mångfald.
- Dela upp datasetet i tränings- och test set.

Vad vi gjorde:

- Vi har skapat en funktion som har tagit en bild och gjort den till en array och ändrat storleken
- Vi har genererat en dataram som innehåller data från varje bild

```python
# skapa en variabel för träningsmappen och valideringsmappen
train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')

# skapa en variabel för batch size och bildstorlek
BATCH_SIZE = 32
IMG_SIZE = (160, 160)
INPUT_SHAPE = IMG_SIZE + (3,)

# skapa en variabel för att lagra träningsdata och valideringsdata
train_dataset = tf.keras.utils.image_dataset_from_directory(train_dir, shuffle=True, batch_size=BATCH_SIZE, image_size=IMG_SIZE)
validation_dataset = tf.keras.utils.image_dataset_from_directory(validation_dir, shuffle=True, batch_size=BATCH_SIZE, image_size=IMG_SIZE)
```

Vi utforskade de identifierade klassnamnen inom datamängderna och lagrade dem för referens. Sedan visualiserade vi några exempelbilder från träningsdatamängden för att få en förståelse för de olika klasserna. I mapperna finns 2 klasser: ['Kirmizi_Pistachio', 'Siirt_Pistachio'].

```
# skapa en variabel för att lagra klassnamn
class_names = train_dataset.class_names
print(f'I mapperna finns {len(class_names)} klasser: {class_names}')
print('\n')
print('Nu ska vi plotta några bilder från träningsdatasetet')

# plotta några bilder från träningsdatasetet
plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```

Eftersom den ursprungliga datauppsättningen inte innehåller en testuppsättning ska vi skapa en. För att göra detta bestämmer vi hur många batcher av data som finns tillgängliga i valideringsuppsättningen med hjälp av tf.data.experimental.cardinality , och flyttar sedan 20% av dem till en testuppsättning.

Vi delade upp valideringsdatamängden och testsdatamängden för att reservera en del av den som testdatamängd.

```
# sätta upp data för att träna modellen
val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)

print('Antal valideringsbatcher: %d' % tf.data.experimental.cardinality(validation_dataset))
print('Antal testbatcher: %d' % tf.data.experimental.cardinality(test_dataset))
```

Vi använde AUTOTUNE för att optimera dataladdningsprocessen. Vi konfigurerar datauppsättningen för att förbättra prestanda

Använd buffrad förladdning för att ladda bilder från disk utan I/O-krascher. Mer information om den här metoden finns i Data Performance Guide .

```
# sätta AUTOTUNE för att förbättra prestanda
AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

Djup inlärning med Convolutional Neural Network (CNN)

Vi skapade en Convolutional Neural Network (CNN) för att dra nytta av bildinformationen. CNN-modellen bestod av flera Conv2D-lager, MaxPooling-lager, Dense-lager och Dropout för att minska risken av overfitting.

Building a CNN Model:

- Designa och implementera en konvolutionellt neuralt nätverk (CNN) arkitektur för bild klassificering.

- Experimentera med olika nätverksarkitekturer, variera antalet lager, filter storlekar och ktiveringsfunktioner.

- Använda tekniker som batchnormalisering och bortfall för regularisering.

Vad vi gjorde:
- Vi har skapat en CNN-modell med 3 Convolutional lager och 4 Dense-lager, och normaliserat data med BatchNormalization. Vi använde också Dropout för att undvika overfitting, och SGD som optimizer.
- Vi har observerade att modellen har en accuracy och loss som är alltid samma värde.

```python
# Vi skapar CNN-mallen med en dedikerad funktion som returnerar den färdiga mallen
def create_model():
    # skapar vi CNN-mallen
    model = tf.keras.models.Sequential()

    # Convolutional Layer 1
    model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=INPUT_SHAPE))       # Convolutional Layer
    model.add(tf.keras.layers.MaxPooling2D((2, 2)))                                                 # Pooling Layer

    # Batch Normalisation
    model.add(tf.keras.layers.BatchNormalization())                                                 # Batch Normalisation

    # Convolutional Layer 2
    model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu'))                                # Convolutional Layer
    model.add(tf.keras.layers.MaxPooling2D((2, 2)))                                                 # Pooling Layer

    # Convolutional Layer 3
    model.add(tf.keras.layers.Conv2D(16, (3, 3), activation='relu'))                                # Convolutional Layer
    model.add(tf.keras.layers.MaxPooling2D((2, 2)))                                                 # Pooling Layer

    # aggiungiamo un layer di flatten e due layer densi
    model.add(tf.keras.layers.Flatten())
```

```python
# calcoliamo la dimensione del layer di flatten
flatten_dim = model.output_shape[1]

# Layer Dense 1: Input Layer + Dropout Layer
model.add(tf.keras.layers.Dense(32, input_dim=flatten_dim, activation='relu', name='input'))                    # Dense Layer
model.add(tf.keras.layers.Dropout(0.4))                                                 # Dropout Layer

# Layer Dense 2: Hidden Layer + Dropout Layer
model.add(tf.keras.layers.Dense(64, activation='relu', name='hidden'))                  # Dense Layer
model.add(tf.keras.layers.Dropout(0.4))                                                 # Dropout Layer
# Layer Dense 3: Hidden Layer + Dropout Layer
model.add(tf.keras.layers.Dense(64, activation='relu', name='hidden2'))                 # Dense Layer
model.add(tf.keras.layers.Dropout(0.4))                                                 # Dropout Layer

# Layer Dense 4: Output Layer
model.add(tf.keras.layers.Dense(1, activation='sigmoid', name='output'))            # Dense Layer
```

Modellen var optimerat med SGD (Stochastic Gradient Descente) genom att att justera vikterna i nätverket för att minimera förlustfunktionen med hänsyn till träningsdata, och en låg inlärningshastigheten bland annat. De olika parametrarna, såsom inlärningshastighet, momentum och klippvärden, påverkar hur snabbt nätverket konvergerar och hur väl det undviker problem som gradientexplosion.

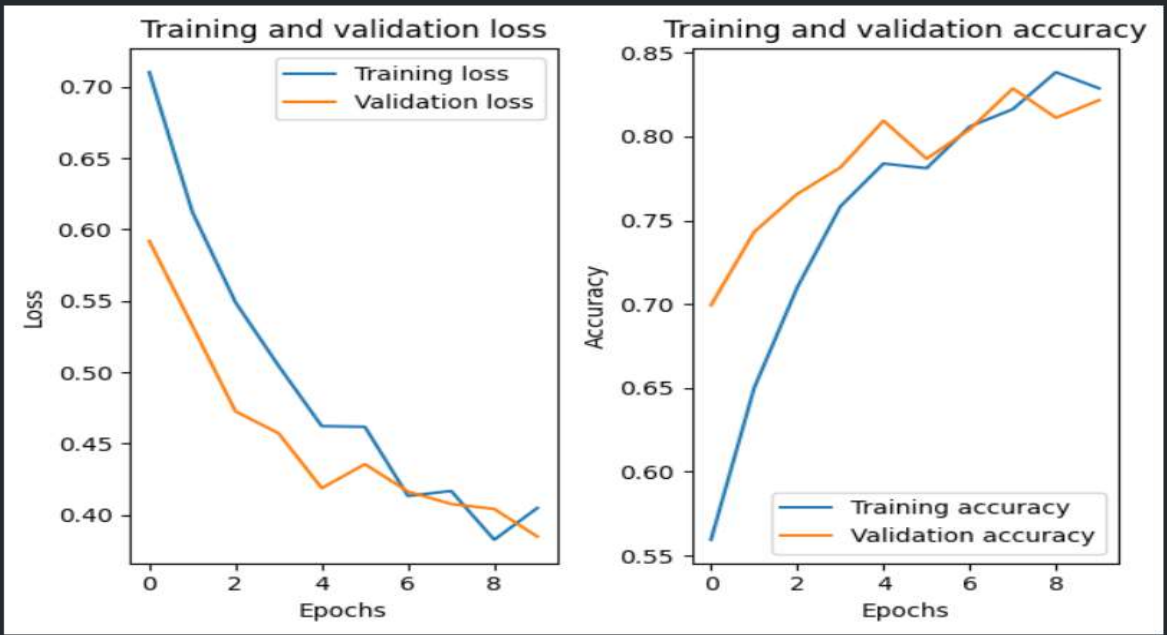
```
# vi optimerar modellen
optimizer = tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.9, nesterov=False, clipnorm=1., clipvalue=0.5)
```

Efter träning, var det möjligt att konstatera loss och accuracy.

```
46/46 [==============================] - 16s 338ms/step - loss: 0.2998 - binary_accuracy: 0.8729
18/18 [==============================] - 7s 357ms/step - loss: 0.4050 - binary_accuracy: 0.8077
-------------------------------------------------
Train Loss:   0.2998
Train Accuracy:   0.8729
-------------------------------------------------
Validation Loss:   0.4050
Validation Accuracy:   0.8077
-------------------------------------------------
```

Vi printar ut resultatet på ett grafiskt sätt.

```
# vi printar ut historiken av träningen och valideringen av modellen
fn.print_history(history, val=True)
```



Djup inlärning med Transfer Learning (VGG16 och MobileNetV2)

Transfer Learning:

- Utforska överföringsinlärning genom att använda förutbildade CNN-modeller som VGG16, ResNet, eller Inception.

- Finjustera de förtränade modellerna på din bilddatauppsättning och utvärdera deras prestanda.

- Jämför resultaten med prestandan hos din specialbyggda CNN-modell.

Vad har vi gjort:
- Vi har laddat ner den förtränade modellen VGG16 och MobileNetV2.
- V har hjälpt modeller med bildförbättringstekniker.
- Vi har använt dataökning för att generera nya roterade och spegelvända bilder för att öka träningsfallen.
- Vi har skalat bilderna till att ha en datamatris mellan -1 och 1 och inte mellan 0 och 255.
- Vi har applicerat förtränade-modellen och sedan een GlobalAveragePooling2D för att sedan bearbeta det med ett lager av neuroner för att få det slutliga resultatet.

https://keras.io/guides/transfer_learning/
https://builtin.com/machine-learning/vgg16
https://towardsdatascience.com/transfer-learning-with-vgg16-and-keras-50ea161580b4
https://github.com/ashushekar/VGG16/blob/master/implementation.py
https://www.tensorflow.org/tutorials/images/transfer_learning?hl=it

Vi skapar en funktion för att generera fler förutbildade och finstämde modeller. Vi använde också dataaugmentering för att öka träningsdataens mångfald och Global Average Pooling 2D för att minska antalat parameter och dimensionaliteten av datan som gör nätverket mer efektiv att träna och köra. Modellen tränades med fokus på att minska loss och öka accuracy.

```
# bildförbättringstekniker för att förbättra modellen
data_augmentation = tf.keras.Sequential([tf.keras.layers.RandomFlip('horizontal'), tf.keras.layers.RandomRotation(0.2),])

# GlobalAveragePooling2D
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()

# prediction layer
prediction_layer = tf.keras.layers.Dense(1)

# skapa listor av loss och accuracy för att kunna plotta dem senare
dictionary_score = {}

# lägga till första modellen i listorna
dictionary_score['initial_model'] = [score_train[0], score_train[1], score_validation[0], score_validation[1]]

# sätt antal epoch vi ska träna modellen
initial_epochs = 10
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs
```

Vi utförde överföringsinlärning med två olika förtränade modeller: VGG16 och MobileNetV2.

Vi skaffade en vgg16 modell som använder små 3x3 konvolutionskärnor, som ger en effektiv inlärning med en låg inlärningshastigheten

```
# sätta namn på modellen
model_name = 'VGG16'

# VGG16
preprocess_VGG16_input = tf.keras.applications.vgg16.preprocess_input
base_model_VGG16 = tf.keras.applications.VGG16(input_shape=INPUT_SHAPE, include_top=False, weights='imagenet')
base_model_VGG16.trainable = False

# skapa modellen
inputs = tf.keras.Input(shape=INPUT_SHAPE)
x = data_augmentation(inputs)
x = preprocess_VGG16_input(x)
x = base_model_VGG16(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)

# sätta in en lägre learning rate
base_learning_rate = 0.0001
```

Vi kompilera modellen mellan att använda Adam optimizer och BinaryCrossentropy(from_logits=True), som är lämplig för binär klassificering (ja/nej eller 0/1) och genereras logit-värden instället för sannilikheter.

```
# kompilera modellen
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Efter träning, var det möjligt att konstatera loss och accuracy.

```
------------------------------------------------
Train Loss:    0.6700
Train Accuracy:    0.6934
------------------------------------------------
```

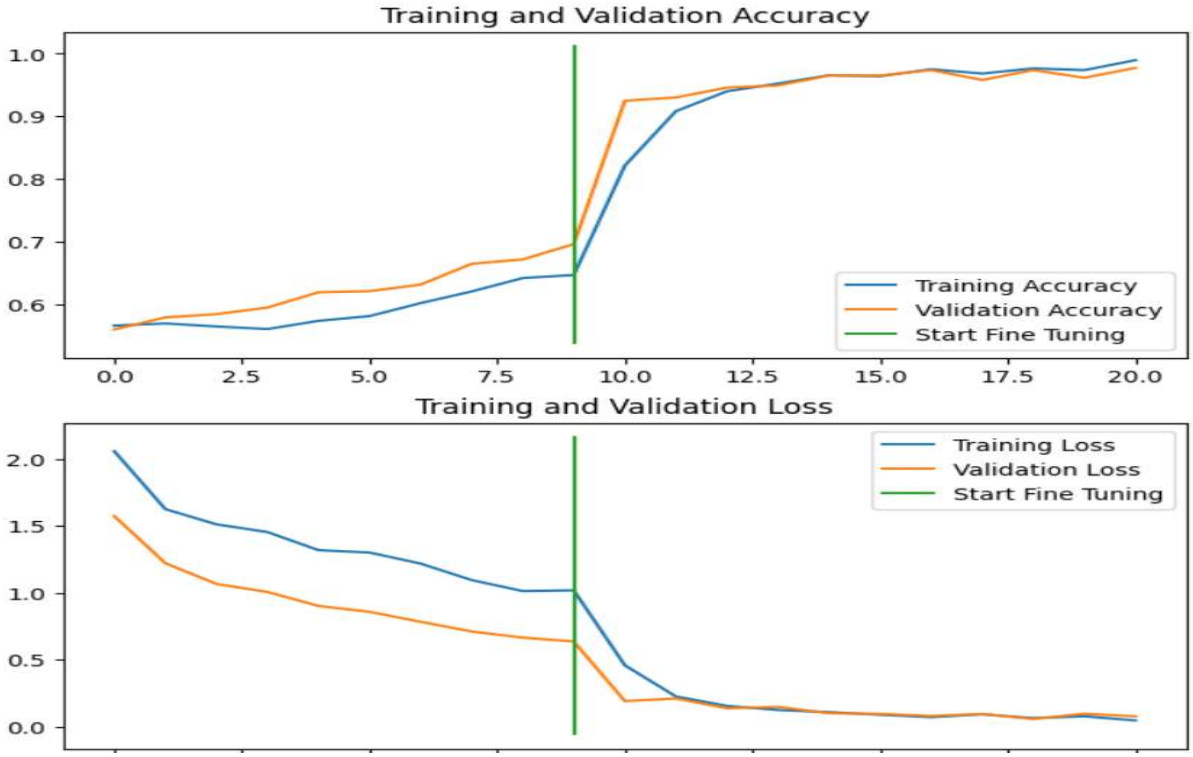
Eftersom resultatet var inte som förväntat, finjustera vi modellen mellan att använda BinaryCrossentropy(from_logits=True), som är lämplig för binär klassificering (ja/nej eller 0/1) och genereras logit-värden instället för sannilikheter. Inlärningshastigheten var minskat för att hjälp finjustering av modellen.

```
# sätta modellens namn
model_name = 'VGG16_fine_tuning'

# sätta base model som trainable
base_model_VGG16.trainable = True

# kompilerar modellen igen
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])

# träna modellen
history_vgg16_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history_vgg16.epoch[-1],
                         validation_data=validation_dataset)
```

Att utvärdera modellen efter finjustering var det möjljgt att konstatera att modellen blev bättre med en låg loss och högre accuracy.

```
------------------------------------------------------------
Train Loss:    0.0182
Train Accuracy:    0.9931
------------------------------------------------------------
```

Vi printar ut en grafik som visar accuracy och loss för VGG16 och den VGG16 finjusterade modellen



MobilNetV2 krävs färre parametrar, körs snabb och får beräkninga jämför med andra djupa nätverksarkitekturer.

Vi skaffade en MobilNetV2 som används data_augmentation, Dropout, har en låg Inlärningshastigheten. Modellen var sättat ihop med Adam optimizers och BinaryCrossentropy(from_logits=True) för binäri klassificering

```
# sätta namn på modellen
model_name = 'MobileNetV2'

# MobileNetV2
preprocess_MobileNetV2_input = tf.keras.applications.mobilenet_v2.preprocess_input
base_model_MobileNetV2 = tf.keras.applications.MobileNetV2(input_shape=INPUT_SHAPE, include_top=False, weights='imagenet')
base_model_MobileNetV2.trainable = False

# skapa modellen
inputs = tf.keras.Input(shape=INPUT_SHAPE)
x = data_augmentation(inputs)
x = preprocess_MobileNetV2_input(x)
x = base_model_MobileNetV2(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)

# sätta in en lägre learning rate
base_learning_rate = 0.0001

# kompilera modellen
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Efter träning, var det möjligt att konstatera att loss och accuracy var inte som förväntat.

```
----------------------------------------------------
Train Loss:    0.3957
Train Accuracy:    0.7825
----------------------------------------------------
```

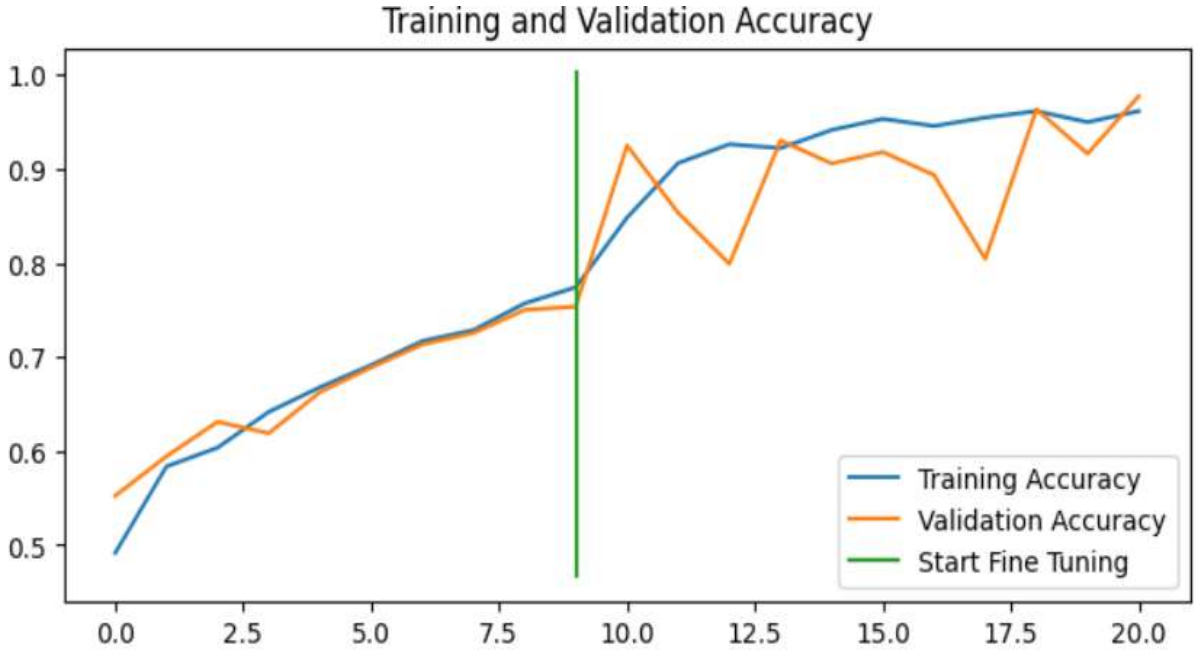
Eftersom resultatet var inte som förväntat, finjustera vi modellen mellan att först fryste vi alla lager och lade till våra egna lager för att anpassa modellen till vårt problem samt att minska Inlärningshastigheten för att hjälp finjustering av modellen.

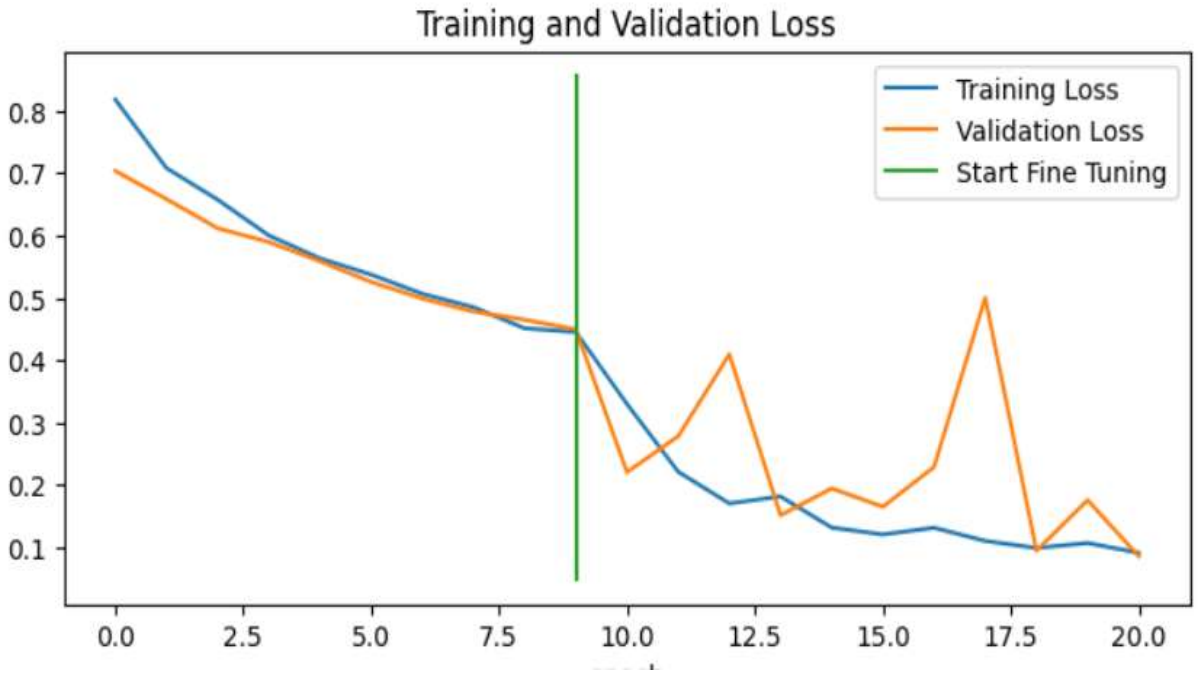
```
# sätta modellens namn
model_name = 'MobileNetV2_fine_tuning'

# sätta base model som trainable
base_model_MobileNetV2.trainable = True

# Finjustera från detta lager och framåt
fine_tune_at = 100

# Frys alla lager före `fine_tune_at`-lagret
for layer in base_model_MobileNetV2.layers[:fine_tune_at]:
  layer.trainable = False

# kompilerar modellen igen
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

Att utvärdera modellen efter finjustering var det möljgt att konstatera att modellen blev bättre med en låg loss och högre accuracy.

```
-------------------------------------------------
Train Loss:    0.0570
Train Accuracy:    0.9841
-------------------------------------------------
```

Vi printar ut en grafik som visar accuracy och loss för MobileNetV2 och den MobileNetV2 finjusterade modellen





Resultat och Utvärdering

Modellutvärdering:

Träna och utvärdera dina CNN-modeller på testdatauppsättningen.
Använd lämpliga utvärderingsmått som noggrannhet, precision, återkallelse och F1 Göra.
Analysera och tolka resultaten för att få insikter i modellens prestanda.

Vad vi gjorde:
- Vi har utvärderat varje modeller med classification report och confusion matrix.
- Vi har jämfört varje modeller med varandra.
- Vi utvecklar en cykel som gör att vi kan utvärdera alla modeller och jämföra resultaten

```
# skapa en loop där vi laddar h5-modellfilen, använder den för att göra förutsägelser och sedan raderar den
# läs filer med tillägget h5 i datamappen
dir = os.listdir(PATH)
dir = [file for file in dir if file.endswith('.h5')]

# vi tar bort elementet model.h5, eftersom det är modellen vi skapade i början för att bearbeta data från excel-filen
dir.remove('model.h5')
```

```
['model_VGG16.h5', 'model_MobileNetV2.h5', 'model_MobileNetV2_fine_tuning.h5', 'model_VGG16_fine_tuning.h5', 'model_cnn.h5']
```

```
# skapa image och label variabler
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
```

Resultaten visar att de djupinlärningsbaserade modellerna, särskilt de med transfer learning, presterade betydligt bättre än logistisk regression. MobileNetV2-fine-tuning-modellen presterade bäst med hög accuracy och låg loss på både tränings- och valideringsdata.

Från utvärderingen av alla modeller framgår det att VGG16-modellen är den som gav den bästa noggrannheten och den bästa förvirringsmatrisen bland alla.

VGG16 finjusterad visade sig dock vara den långsammaste av alla.
Modellen med bäst prestanda visade sig vara den finjusterade MobileNetV2.

Model_cnn:

```
4/4 [==============================] - 2s 476ms/step - loss: 0.3759 - binary_accuracy: 0.8203
==================================================
model_cnn.h5
==================================================
Test Loss:   0.3759
Test Accuracy:   0.8203
--------------------------------------------------
classification report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        18
           1       0.44      1.00      0.61        14

    accuracy                           0.44        32
   macro avg       0.22      0.50      0.30        32
weighted avg       0.19      0.44      0.27        32

--------------------------------------------------
confusion matrix
[[ 0 18]
 [ 0 14]]
```

Model_VGG16:

```
4/4 [==============================] - 30s 8s/step - loss: 0.5571 - accuracy: 0.7188
=================================================
model_VGG16.h5
=================================================
Test Loss:   0.5571
Test Accuracy:   0.7188
-------------------------------------------------
classification report
              precision    recall  f1-score   support

           0       0.89      0.94      0.92        18
           1       0.92      0.86      0.89        14

    accuracy                           0.91        32
   macro avg       0.91      0.90      0.90        32
weighted avg       0.91      0.91      0.91        32

-------------------------------------------------
confusion matrix
[[17  1]
 [ 2 12]]
```

Model_MobilNetV2:

```
4/4 [==============================] - 3s 518ms/step - loss: 0.3958 - accuracy: 0.7266
=================================================
model_MobileNetV2.h5
=================================================
Test Loss:   0.3958
Test Accuracy:   0.7266
-------------------------------------------------
classification report
              precision    recall  f1-score   support

           0       0.86      1.00      0.92        18
           1       1.00      0.79      0.88        14

    accuracy                           0.91        32
   macro avg       0.93      0.89      0.90        32
weighted avg       0.92      0.91      0.90        32

-------------------------------------------------
confusion matrix
[[18  0]
 [ 3 11]]
```

Model_VGG16_Fine_tuning

```
4/4 [==============================] - 33s 8s/step - loss: 0.1336 - accuracy: 0.9766
=================================================
model_VGG16_fine_tuning.h5
=================================================
Test Loss:   0.1336
Test Accuracy:   0.9766
-------------------------------------------------
classification report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      1.00      1.00        14

    accuracy                           1.00        32
   macro avg       1.00      1.00      1.00        32
weighted avg       1.00      1.00      1.00        32

-------------------------------------------------
confusion matrix
[[18  0]
 [ 0 14]]
```

Model_MobilNetV2_Fine_tuning

```
4/4 [==============================] - 3s 506ms/step - loss: 0.0370 - accuracy: 1.0000
================================================
model_MobileNetV2_fine_tuning.h5
================================================
Test Loss:   0.0370
Test Accuracy:   1.0000
------------------------------------------------
classification report
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        18
           1       0.93      1.00      0.97        14

    accuracy                           0.97        32
   macro avg       0.97      0.97      0.97        32
weighted avg       0.97      0.97      0.97        32

------------------------------------------------
confusion matrix
[[17  1]
 [ 0 14]]
```

Slutsatser

Detta projekt illustrerar framgången med djupinlärningstekniker för bildklassificering. Genom att använda CNN och transfer learning kunde vi uppnå hög prestanda med hög accuracy och låg loss. Resultaten är lovande och visar potentialen för användning av liknande tekniker för att lösa problem med bildklassificering. Detta kan vara särskilt värdefullt inom områden som kvalitetskontroll och livsmedelsindustrin där snabb och exakt klassificering av objekt är avgörande.