

## CS-C3240 Machine Learning D Project

### 1. Introduction

Shipping is one of the most common ways to transport products from one country to another. It is essential and important to monitor and analyze shipping activities, and nowadays satellite imagery in combination with machine learning can be a powerful tool in keeping track of ships and their activities cost-effectively.

In this project we aim to develop a machine learning model using convolutional neural networks to automatically detect ships from satellite images. In practice, this means classifying the image either as containing a ship or not containing one. Convolutional Neural Networks (CNNs) have proven to be powerful in image recognition tasks due to their ability to automatically learn spatial hierarchies of features from raw pixel data, making them ideal for analyzing complex visual patterns. Similarly, K-Nearest Neighbors (KNN) is a simple yet robust algorithm well suited for image classification.

In this report, we will first discuss our goals and dataset used as well as preprocessing and feature selection done to the data. Next, we will introduce both machine learning methods used as well as training and validation in relation to these methods. In results -chapter we discuss how well the models performed, and finally we conclude how well these methods performed for our task.

### 2. Problem Formulation

The goal of this project is to develop a machine learning model that detects and classifies ships accurately from satellite imagery using CNNs. This problem falls under the supervised learning category. Specifically this is a classification problem.

#### Data

For the data, we used the dataset ‘Ships in Satellite images’ found on Kaggle.com (<https://www.kaggle.com/datasets/rhammell/ships-in-satellite-imagery/data>), which consists of eight satellite images and 4000 training images, which are divided into two categories (1000 images containing ships and 3000 images of locations without ships). Each image is an 80x80 Rgb image cropped from satellite imagery either from San Francisco Bay or San Pedro Bay from California. The dataset is formatted in a zipped directory containing .png images and a JSON file with labels and metadata. Each image is labeled, and the filenames contain metadata (scene ID, longitude, and latitude). For our task, we only needed the training data.

### 3. Methods

#### Data Preprocessing

The data needs to be preprocessed using normalization. We may also need to augment (e.g flip or rotate) to make the training more robust. Labels will be converted to numerical values 1 for ship and 0 for no ship. We will normalize the data to range  $[0,1]$  dividing each pixel by 255.

**Normalization:** Each pixel value will be normalized to the range  $[0, 1]$  by dividing by 255. This ensures all features are on the same scale and improves training efficiency.

**Data Augmentation:** To increase training data diversity and improve model robustness, techniques like flipping or rotating images were utilized.

**Label Encoding:** Class labels ("ship" and "no ship") were converted to numerical values (e.g., 1 for ship, 0 for no ship) for compatibility with the model.

### **Feature selection**

There is no need for additional feature selection since the dataset already had the separation done with filenames of training images being encoded with either 1 or 0 (the ships and images containing no ships), and separating these into their own folders was a trivial task. The focus will be on using CNN to extract hierarchical features from the images. Also Convolutional Neural Networks (CNNs) are designed to automatically extract relevant features from raw image data.

### **Model choice 1: Convolutional Neural Network (CNN)**

We have chosen convolutional neural networks for our first model because it is a method designed for image-based data and performs well in classification problems. CNNs use convolutional layers to automatically learn hierarchical features from images, making them particularly well-suited for visual data. Convolutional layers automatically learn filters that capture spatial features like edges, shapes, and textures from images. These layers are stacked to build increasingly complex representations, allowing the network to distinguish ships from non-ship images.

### **CNN Loss function: Cross-Entropy Loss**

For our loss function for CNN, we have chosen to use cross-entropy loss, or log loss, function. It measures the difference between the predicted probability distribution and the true label distribution. Minimizing this loss function during training encourages the model to make accurate class predictions.

### **Model choice 2: k-Nearest Neighbors (KNN)**

For our second model we chose k-nearest neighbors -algorithm (KNN). KNN is a supervised learning method and one of the simplest image classification algorithms. We are using KNN as an alternative method for classifying ships in satellite images. KNN works by storing the entire training dataset and making predictions based on the closest data points (neighbors) to a given test example.

The basic idea behind KNN is to classify a new data point by comparing it to the stored training examples and finding the “k” closest neighbors, with k being a variable. The closeness is usually measured using a distance metric, such as Euclidean distance (as we used in this project), which calculates the straight-line distance between two points in a multi-dimensional space. For classification, the class label of the new point is determined by a majority vote among its nearest neighbors. In this case, if the majority of the nearest neighbors are labeled as "ship," the new image will also be classified as containing a ship; otherwise, it will be classified as "no ship."

The only thing we will add to this model is that we will need to use feature selection. This is because using pixel values as features may result in high dimensional feature space where distance calculation will be meaningless. To tackle this problem we will do a dimensionality reduction using Principal Component Analysis (PCA). Otherwise for KNN the same data preprocessing that was used with CNN is sufficient.

Loss function was not needed for KNN. This is because the algorithm memorizes the training data and makes predictions based on the distances between the test data and the training data.

For choosing the optimal k-value we used cross validation. Too big k-value might lead to overfitting and too small k might lead to underfitting. We found two nearest neighbors being the ideal k-value for our task.

## Model Validation

The dataset was split into three sets:

- **Training Set (80%):** Used to train the CNN model.
- **Validation Set (10%):** Used to monitor model performance during training and adjust hyperparameters (like learning rate) to prevent overfitting.
- **Test Set (10%):** Used for final evaluation of the model's generalization performance on unseen data.

80% training set provides a robust dataset for the model to learn from. Given that neural networks, such as CNNs, require a significant amount of data to learn complex patterns. The 10% validation set is critical for tuning hyperparameters and monitoring model performance during training. By reserving 10% of the data for validation, we can evaluate the model's performance on unseen data, helping to prevent overfitting. This portion allows for adjustments to be made based on validation accuracy and loss. The 10% test set, comprising the final 10%, serves as an unbiased evaluation of the model's performance after training and validation. This is essential for assessing how well the model can generalize to new, unseen data.

We ran the training for the CNN model multiple times through the dataset to minimize the errors while trying to avoid overfitting. Our code included a method for pausing the training as results stopped improving, thus giving us ideal weights for our model. We found eleven rotations to be the optimal number of rotations for our task.

- **Evaluation Metrics:** To evaluate the model's performance, we used the following metrics:
- **Accuracy:** Measures the percentage of correctly classified images out of the total images.
- **Precision:** The proportion of true positives (correctly classified ship images) out of all positive predictions (all images predicted as ships).
- **Recall (Sensitivity):** The proportion of true positives out of all actual ship images in the dataset.
- **F1 Score:** The harmonic mean of precision and recall, which provides a balanced measure when there is an uneven class distribution (e.g., more "no ship" images than "ship" images).

## 4. Results

### Training and Validation Errors for CNN

As the model ran through the dataset in multiple loops (epochs), the training accuracy steadily improved from 91.37% (epoch 1) to over 92.68% (epoch 11). Meanwhile, validation accuracy ranged from 93.50% to 95.50%.

The training loss decreased consistently throughout the epochs, from 0.2145 (epoch 1) to 0.1749 (epoch 11). On the other hand, validation loss fluctuates around 0.12–0.15. The steady improvement in training loss combined with only small changes in validation loss is a positive indicator of convergence.

The classification report shows strong performance, with particularly high precision (0.99) and recall (0.97) for "No Ship" images. For "Ship" images, precision is slightly lower at 0.91 but recall is 0.98, which indicates the model is quite sensitive in detecting ships, even if it occasionally predicts a ship when there isn't one.

### Training and Validation Errors for KNN

The findings are similar to CNN here. We have high values for accuracy, precision, recall and F1 indicating that the model finds ships well and there are minimal false positives and false negatives. Precision is a little bit better here in KNN than in CNN. It seems KNN performed much better than expected. We believe we found the optimal KNN for this model.

### Final model

For the final model we chose CNN, because it performed better and with a little bit of tuning, we can get even better performance. CNN is also a more powerful tool in image classification in general because of its ability to learn complex hierarchical patterns directly from the data. If our image classification problem was more complex than this, we believe that CNN would have had much better performance than KNN.

The test error can be derived from the test accuracy:

- **Test Error** =  $1 - \text{Test Accuracy}$
- **Test Error** =  $1 - 0.9700 = 0.0300$  (or 3.00%)
- **Test Accuracy**: 97.00%
- **Test Loss**: 0.0971

## 5. Conclusion

In this report, we explored the application of machine learning techniques, specifically Convolutional Neural Networks (CNN) and k-Nearest Neighbors (KNN), for the task of ship detection from satellite imagery. Our findings indicate that both methods are viable for this classification problem, with CNN generally outperforming KNN across most evaluation metrics. However, KNN exhibited a higher precision score, showcasing its ability to minimize false positives in certain scenarios.

We found both CNN and KNN methods to be suitable for the task of detecting ships from satellite imagery. While CNN performed slightly better for most of the used metrics, KNN seemed to produce better results in precision score. The high validation accuracy suggests that the model generalizes well to unseen data, avoiding significant overfitting. However, small fluctuations in the validation loss suggest that further fine-tuning (e.g., adjusting learning rate or introducing regularization) could reduce minor overfitting effects. Also a better method for finding optimal epochs is required. The stopper we used sometimes stops the training too soon resulting in underfitting.

In conclusion, while our current results suggest that the problem was solved satisfactorily, further fine tuning of CNN would be required for even better performance. KNN can be used in simple image classification tasks, but high dimensionality of this type of data is a limiting factor for the model even if PCA or other dimensionality reduction methods were used.

## 6. References

Wikipedia contributors. (2024). *Convolutional neural network*. Wikipedia, The Free Encyclopedia. Retrieved from [https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network)

Aalto University. (2024). *Machine Learning D Course Slides*. Department of Computer Science, Aalto University.

Hammell, R. (2018). *Ships in Satellite Imagery Dataset*. Kaggle. Retrieved from <https://www.kaggle.com/datasets/rhammell/ships-in-satellite-imagery>

LaViale, Trevor. (2023). Deep dive on KNN: Understanding and implementing the k-nearest neighbors algorithm. Retrieved from <https://arize.com/blog-course/knn-algorithm-k-nearest-neighbor/>

<https://www.tensorflow.org/tutorials/images/cnn>

## 7. Appendix

13/13 - 1s - loss: 0.0971 - accuracy: 0.9700 - 573ms/epoch - 44ms/step

Test Accuracy: 0.9700

Test Loss: 0.0971

13/13 [=====] - 1s 48ms/step

Classification Report:

	precision	recall	f1-score	support
No Ship	0.99	0.97	0.98	300
Ship	0.91	0.98	0.94	100
accuracy			0.97	400
macro avg	0.95	0.97	0.96	400
weighted avg	0.97	0.97	0.97	400

Epoch 1/50

100/100 [=====] - 30s 297ms/step - loss: 0.2145 - accuracy: 0.9131 - val\_loss: 0.1564 - val\_accuracy: 0.9525

Epoch 2/50

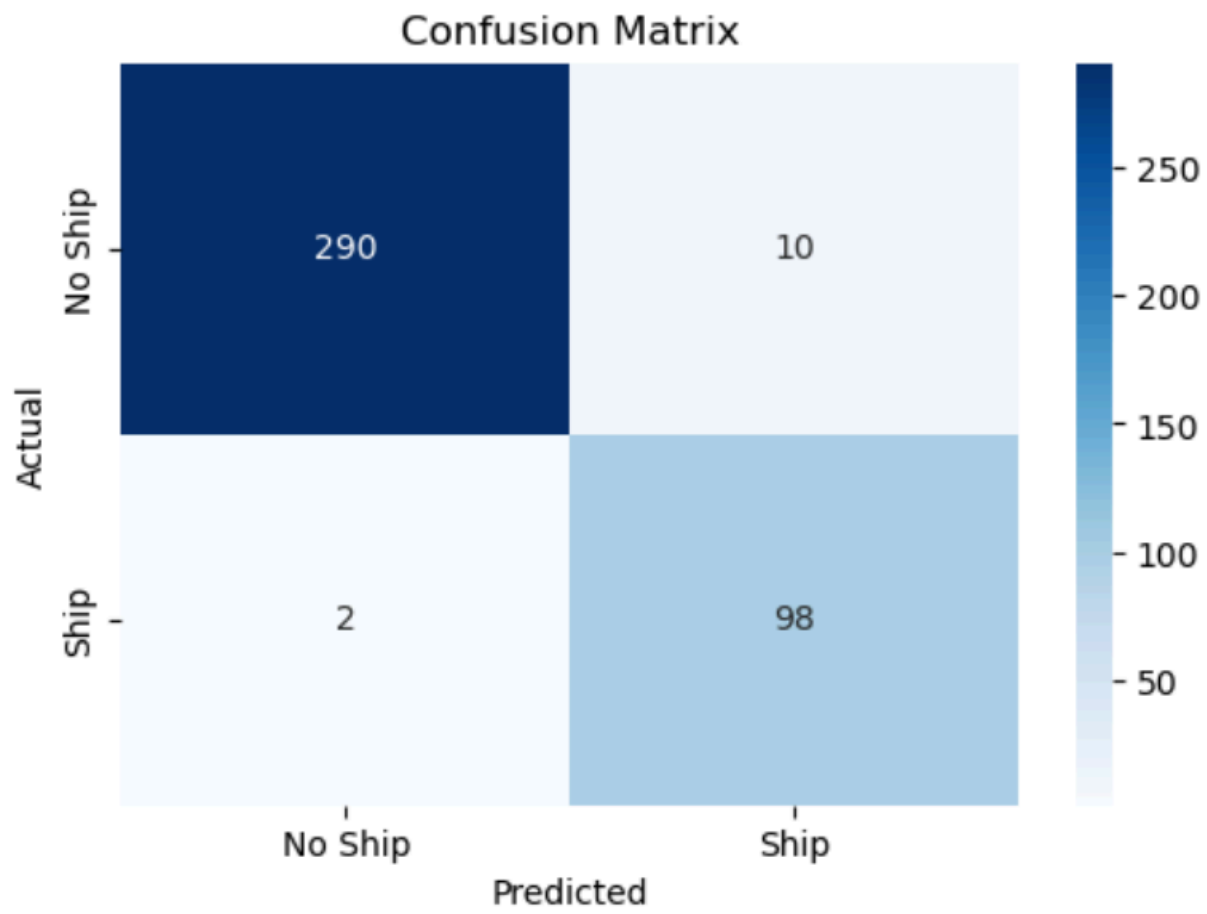
100/100 [=====] - 26s 265ms/step - loss: 0.1999 - accuracy: 0.9178 - val\_loss: 0.1480 - val\_accuracy: 0.9500

Epoch 3/50

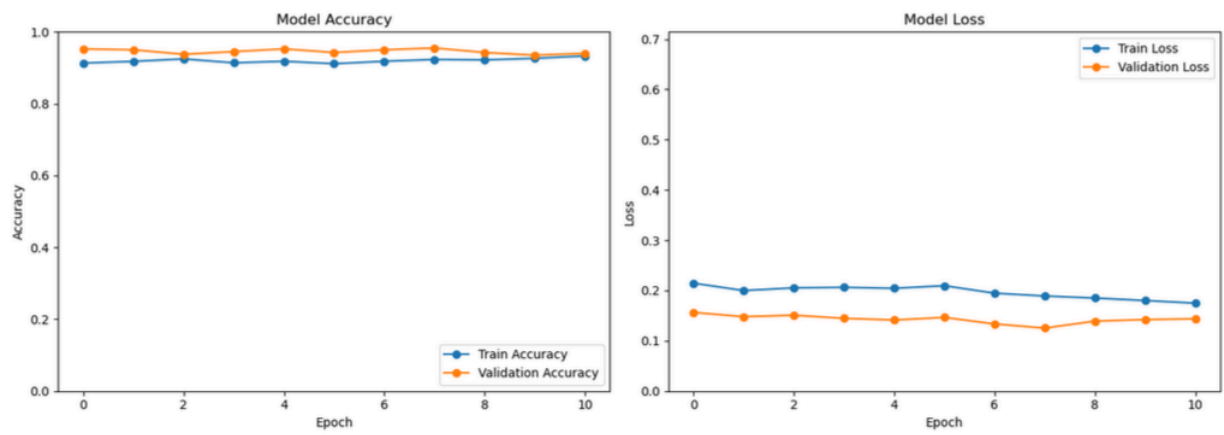
100/100 [=====] - 30s 304ms/step - loss: 0.2053 - accuracy: 0.9247 -

Epoch 4/50  
100/100 [=====] - 32s 323ms/step - loss: 0.2063 - accuracy: 0.9137 -  
val\_loss: 0.1446 - val\_accuracy: 0.9450  
Epoch 5/50  
100/100 [=====] - 27s 273ms/step - loss: 0.2044 - accuracy: 0.9181 -  
val\_loss: 0.1413 - val\_accuracy: 0.9525  
Epoch 6/50  
100/100 [=====] - 26s 256ms/step - loss: 0.2095 - accuracy: 0.9112 -  
val\_loss: 0.1465 - val\_accuracy: 0.9425  
Epoch 7/50  
100/100 [=====] - 29s 289ms/step - loss: 0.1947 - accuracy: 0.9181 -  
val\_loss: 0.1335 - val\_accuracy: 0.9500  
Epoch 8/50  
100/100 [=====] - 26s 265ms/step - loss: 0.1891 - accuracy: 0.9231 -  
val\_loss: 0.1253 - val\_accuracy: 0.9550  
Epoch 9/50  
100/100 [=====] - 29s 290ms/step - loss: 0.1850 - accuracy: 0.9219 -  
val\_loss: 0.1391 - val\_accuracy: 0.9425  
Epoch 10/50  
100/100 [=====] - 27s 266ms/step - loss: 0.1801 - accuracy: 0.9262 -  
val\_loss: 0.1422 - val\_accuracy: 0.9350  
Epoch 11/50  
100/100 [=====] - 28s 283ms/step - loss: 0.1749 - accuracy: 0.9328 -  
val\_loss: 0.1436 - val\_accuracy: 0.9400

CNN confusion matrix



CNN



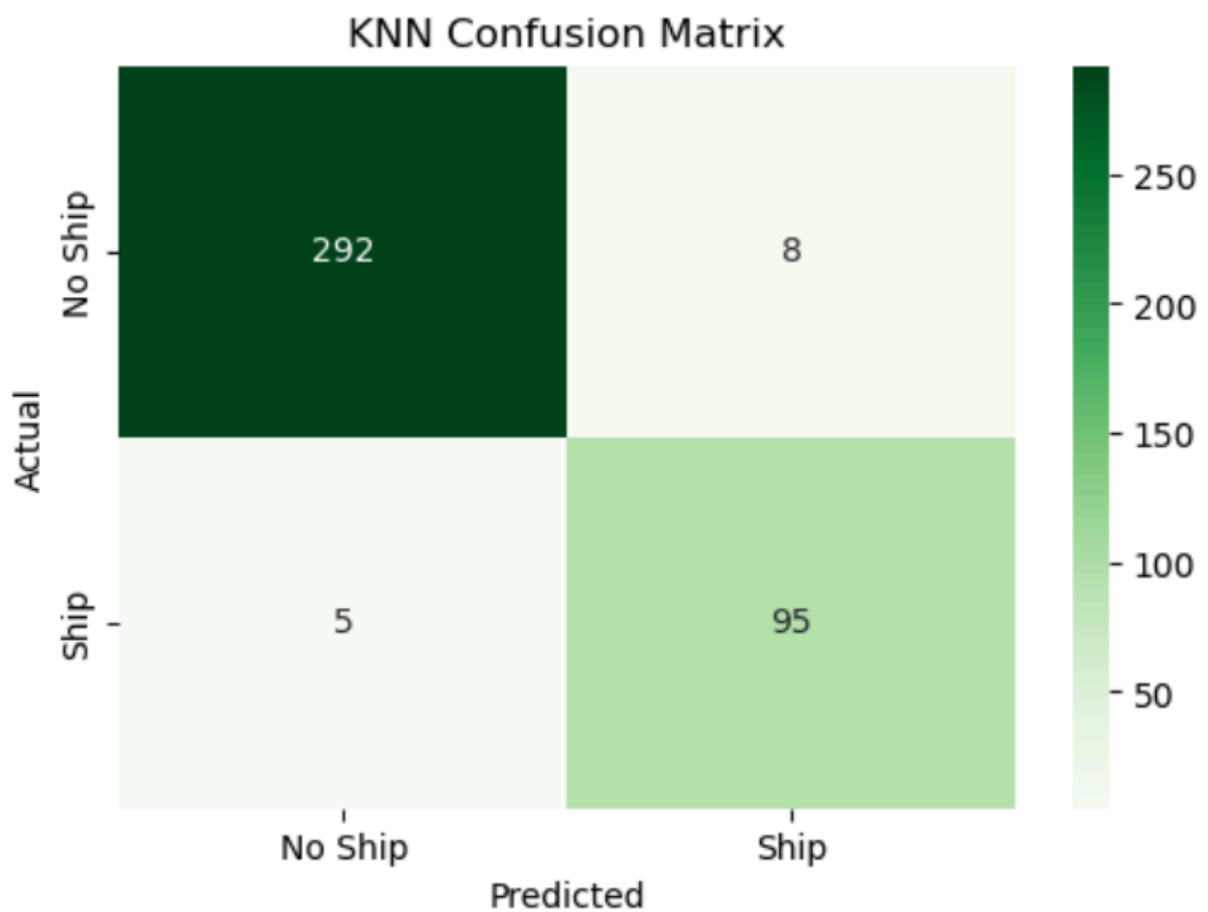
### KNN Classification Report:

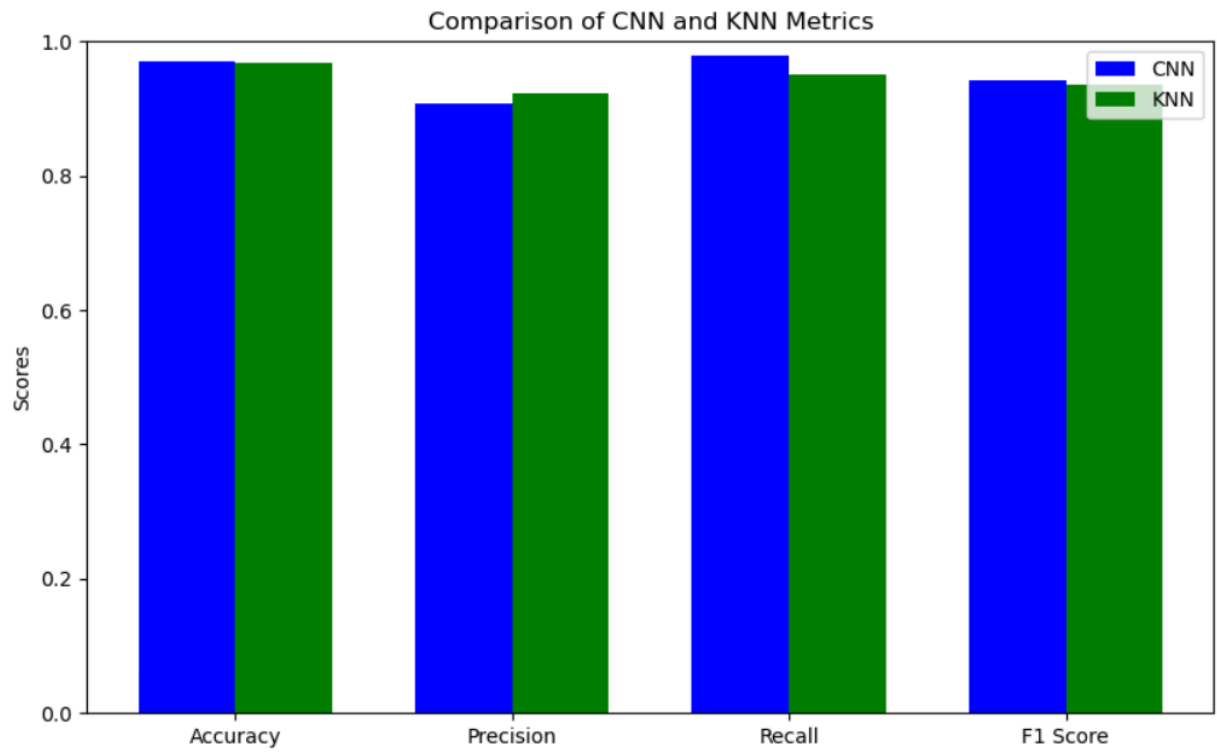
	precision	recall	f1-score	support
No Ship	0.98	0.97	0.98	300
Ship	0.92	0.95	0.94	100
accuracy			0.97	400
macro avg	0.95	0.96	0.96	400
weighted avg	0.97	0.97	0.97	400

Explained variance by 50 components: 0.84

Best KNN parameters: {'metric': 'manhattan', 'n\_neighbors': 2, 'weights': 'uniform'}

Best KNN cross-validation accuracy: 0.9589





# projekti

October 9, 2024

```
[2]: ## Koodit tänne
import os
import numpy as np
import pandas as pd
import json
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, \
    f1_score, classification_report, \
    confusion_matrix
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, \
    Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder, StandardScaler
from tensorflow.keras.callbacks import EarlyStopping

data_dir = "data"

def load_images(data_dir):
    images = []
    labels = []

    for label, folder in enumerate(["no_ship", "ship"]):
        folder_path = os.path.join(data_dir, folder)
        for img_path in os.listdir(folder_path):
            img_path = os.path.join(folder_path, img_path)
```

```

        img = Image.open(img_path).convert('RGB')
        img = img.resize((80, 80))
        img_array = np.array(img)
        images.append(img_array)
        labels.append(label)

    return np.array(images), np.array(labels)

# load images
images, labels = load_images(data_dir)

# normalize the pixel values to [0,1]
images = images / 255.0

# convert labels to categorical
labels = to_categorical(labels, num_classes=2)

# First split: Train (80%) and Temp (20%)
X_train, X_temp, y_train, y_temp = train_test_split(
    images,
    labels,
    test_size=0.2,
    random_state=42,
    stratify=labels
)

# Second split: Validation (10%) and Test (10%) from Temp
X_val, X_test, y_val, y_test = train_test_split(
    X_temp,
    y_temp,
    test_size=0.5,
    random_state=42,
    stratify=y_temp
)

# spilt for KNN
y_train_knn = y_train.argmax(axis=1)
y_val_knn = y_val.argmax(axis=1)
y_test_knn = y_test.argmax(axis=1)

# Now we have have:
# - X_train, y_train for training the CNN
# - X_val, y_val for validating the CNN
# - X_test, y_test for testing the CNN

```

```

# - y_train_knn, y_val_knn, y_test_knn for KNN

# data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    horizontal_flip=True,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2
)

# Fit the data generator on the training data
datagen.fit(X_train)

# Build the CNN model
model = Sequential()

# First Conv Block
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(80, 80, 3)))
model.add(MaxPooling2D((2, 2)))

# Second Conv Block
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Third Conv Block
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Flatten and Dense Layers
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5)) # Dropout for regularization
model.add(Dense(2, activation='softmax')) # 2 classes

# Display the model architecture
model.summary()

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Define data augmentation
datagen = ImageDataGenerator(

```

```

        rotation_range=20,
        horizontal_flip=True,
        zoom_range=0.2,
        width_shift_range=0.2,
        height_shift_range=0.2
    )

    # Fit the data generator on the training data
    datagen.fit(X_train)

    # Train the model (without stopper)
    history = model.fit(
        datagen.flow(X_train, y_train, batch_size=32),
        epochs=11,
        validation_data=(X_val, y_val)
    )

    # stopper for optimal epoch
    # early_stopping = EarlyStopping(
    #     monitor='val_loss', # or 'val_accuracy'
    #     patience=3,         # Stop training after 3 epochs without improvement
    #     restore_best_weights=True # Restore the weights from the best-performing
    #                             ↪ epoch
    # )

    # Train the model
    # history = model.fit(
    #     datagen.flow(X_train, y_train, batch_size=32),
    #     epochs=50, # You can set a large number of epochs, say 50 or 100
    #     validation_data=(X_val, y_val),
    #     callbacks=[early_stopping]
    # )

    # Evaluate the model on the test set
    test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
    print(f"\nTest Accuracy: {test_acc:.4f}")
    print(f"Test Loss: {test_loss:.4f}")

    # Generate classification report
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_true = np.argmax(y_test, axis=1)

    print("\nClassification Report:")
    print(classification_report(y_true, y_pred_classes, target_names=['No Ship',
    ↪ 'Ship']))

    # Confusion Matrix

```

```

cm = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Ship', 'Ship'], yticklabels=['No Ship', 'Ship'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()

# Plot training & validation accuracy values
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy', marker='o')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marker='o')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.ylim([0, 1])
plt.legend(loc='lower right')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.ylim([0, max(history.history['loss']) + 0.5])
plt.legend(loc='upper right')

plt.tight_layout()
plt.show()

# ----- K-Nearest Neighbors (KNN) Model -----

# Flatten images for KNN
X_train_knn = X_train.reshape(X_train.shape[0], -1)
X_val_knn = X_val.reshape(X_val.shape[0], -1)
X_test_knn = X_test.reshape(X_test.shape[0], -1)

# Feature Scaling
scaler = StandardScaler()

```

```

X_train_knn = scaler.fit_transform(X_train_knn)
X_val_knn = scaler.transform(X_val_knn)
X_test_knn = scaler.transform(X_test_knn)

# Dimensionality Reduction using PCA
pca = PCA(n_components=50, random_state=42) # Adjust n_components as needed
X_train_pca = pca.fit_transform(X_train_knn)
X_val_pca = pca.transform(X_val_knn)
X_test_pca = pca.transform(X_test_knn)

print(f"Explained variance by 50 components: {np.sum(pca.
    ↪ explained_variance_ratio_):.2f}")

# Combine training and validation sets for KNN training
X_knn_train = np.vstack((X_train_pca, X_val_pca))
y_knn_train = np.concatenate((y_train_knn, y_val_knn))

# Hyperparameter Tuning for KNN using GridSearchCV
param_grid = {
    'n_neighbors': list(range(1, 31)),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

knn = KNeighborsClassifier()

grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_knn_train, y_knn_train)

print(f"Best KNN parameters: {grid_search.best_params_}")
print(f"Best KNN cross-validation accuracy: {grid_search.best_score_:.4f}")

# Best KNN model
best_knn = grid_search.best_estimator_

# Evaluate KNN on Test Set
y_pred_knn = best_knn.predict(X_test_pca)
knn_accuracy = accuracy_score(y_test_knn, y_pred_knn)
knn_precision = precision_score(y_test_knn, y_pred_knn)
knn_recall = recall_score(y_test_knn, y_pred_knn)
knn_f1 = f1_score(y_test_knn, y_pred_knn)

print("\nKNN Classification Report:")
print(classification_report(y_test_knn, y_pred_knn, target_names=['No Ship',
    ↪ 'Ship']))

```

```

# Confusion Matrix for KNN
cm_knn = confusion_matrix(y_test_knn, y_pred_knn)
plt.figure(figsize=(6, 4))
sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Greens', xticklabels=['No Ship', 'Ship'], yticklabels=['No Ship', 'Ship'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('KNN Confusion Matrix')
plt.show()

# Plot KNN Evaluation Metrics
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
cnn_metrics = [
    accuracy_score(y_true, y_pred_classes),
    precision_score(y_true, y_pred_classes),
    recall_score(y_true, y_pred_classes),
    f1_score(y_true, y_pred_classes)
]
knn_metrics = [knn_accuracy, knn_precision, knn_recall, knn_f1]

x = np.arange(len(metrics))
width = 0.35

plt.figure(figsize=(10, 6))
plt.bar(x - width/2, cnn_metrics, width, label='CNN', color='blue')
plt.bar(x + width/2, knn_metrics, width, label='KNN', color='green')

plt.ylabel('Scores')
plt.title('Comparison of CNN and KNN Metrics')
plt.xticks(x, metrics)
plt.ylim([0, 1])
plt.legend()
plt.show()

```

2024-10-09 15:20:03.898569: I tensorflow/core/platform/cpu\_feature\_guard.cc:182]

This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.

To enable the following instructions: SSE4.1 SSE4.2 AVX, in other operations, rebuild TensorFlow with the appropriate compiler flags.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 78, 78, 32)	896
max_pooling2d (MaxPooling2	(None, 39, 39, 32)	0