1. Introduction – Project Aim

We worked on the Fashion MNIST image dataset, which is available within the Keras library, using deep learning and machine learning algorithms. We then evaluated the effectiveness of our model using evaluation metrics.

2. Material – Method

a. The image dataset used:

The image dataset we used is the Fashion MNIST available in the Keras library.

b. Machine learning or deep learning algorithms used:

- **Deep learning algorithm:** We first used the ANN algorithm. Thus, we trained the model using TensorFlow's Keras API.
- **Machine learning algorithm:** Secondly, we used the KNN algorithm. This algorithm determines which class label a data point belongs to by looking at the majority of the labels of its nearest neighbors. This algorithm is also an example of a supervised learning algorithm.

c. Evaluation Metrics:

- **Accuracy:** This metric measures the ratio of correctly classified samples to the total samples.
- **Precision:** This metric evaluates the ratio of true positive results among the results predicted as positive by the model. It is calculated with the average='weighted' parameter to account for class imbalances.
- **Recall:** This metric evaluates the ratio of true positive results among the actual positive samples.
- **F1 Score:** This metric is the harmonic mean of precision and recall and provides a balance between the two metrics.

3. Information about the dataset:

The Fashion MNIST dataset is an image dataset containing various clothing categories and is frequently used in image classification problems. This dataset consists of a total of 70,000 examples.

Features:

- **Image Dimensions:** Each image is 28x28 pixels in size.
- **Pink Scale:** Each pixel has a value between 0 and 255.

Data Quantity:

- **Training dataset:** 60,000 images.
- **Test dataset:** 10,000 images.

Classes:

There are a total of 10 classes in the dataset. Each class represents a different fashion item. The classes and their corresponding labels are:

- **Label 0:** T-shirt/Top
- **Label 1:** Trouser
- **Label 2:** Pullover
- **Label 3:** Dress
- **Label 4:** Coat
- **Label 5:** Sandal
- **Label 6:** Shirt
- **Label 7:** Sneaker
- **Label 8:** Bag
- **Label 9:** Ankle Boot

MODELS USED

1.ANN- Artificial Neural Networks

Structure

Input Layer: Flattens 28x28 images.

Hidden Layer: 128 neurons, ReLU activation function.

Output Layer: 10 neurons, Softmax activation function.

```
model = Sequential([

Flatten(input_shape=(28, 28)), #input layer flattens the data

Dense(128, activation='relu'), #hidden layer - fully connected

Dense(10, activation='softmax') #output layer

])
```

Sequential Class:

In Keras, the Sequential model type allows us to add layers one after another. Each layer takes the output of the previous layer as input.

Input Layer (Flatten Layer):

Converts 2D image data into 1D vectors, making it suitable for further processing by subsequent layers.**Hidden Layer:**

Contains 128 neurons with ReLU activation function. The ReLU function sets negative values to zero and keeps positive values unchanged. This helps the model learn and map abstract features to a higher-dimensional space, allowing it to recognize complex patterns.

Output Layer:

Contains 10 neurons, corresponding to the 10 categories in the dataset. Uses the Softmax activation function, which converts the output into a probability distribution. Each class gets a probability value between 0 and 1, and the sum of all probabilities equals 1. This helps in determining the class with the highest probability as the model's prediction.

Explanation:

Flatten Layer: Converts 2D images (28x28 pixels) into 1D vectors (784 elements) to be processed by the dense layers.

Dense Layer: Fully connected layer where each neuron is connected to all neurons in the previous layer.

128 Neurons with ReLU Activation: Helps the model learn complex patterns by mapping features to a higher-dimensional space.

10 Neurons with Softmax Activation: Provides a probability distribution over the 10 classes, allowing the model to predict the class with the highest probability.

-This architecture helps the model understand and classify the clothing images in the Fashion MNIST dataset effectively.

2.KNN - K-Nearest Neighbors Model

Structure:

n_neighbors: 3

n_neighbors: This parameter specifies the number of nearest neighbors to consider.

In this case, 3 neighbors are used.

KNN is a simple, supervised machine learning algorithm used for classification tasks. It predicts the class of a data point based on the classes of its nearest neighbors in the feature space.

Reshape the Data:

The image data is converted from 2D (28x28 pixels) to 1D (784 elements) to be compatible with the KNN algorithm.**Training:**

The model stores the training data and uses it to calculate the distance between each test point and all training points during prediction.

Predictions:

For each test point, the model identifies the 3 closest training points and assigns the most common class among them as the predicted class.

Evaluation Metrics:

Accuracy: The proportion of correct predictions out of all predictions.

Precision: The proportion of true positive predictions out of all positive predictions.

Recall: The proportion of true positive predictions out of all actual positives.

F1 Score: The harmonic mean of precision and recall, providing a balance between the two.

EXPERIMENT RESULTS

1. ANN - Artificial Neural Network Model

```
#training of the model
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

```
Epoch 1/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.5007 -
accuracy: 0.8248 - val_loss: 0.4109 - val_accuracy: 0.8515
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3760 -
```

```
accuracy: 0.8664 - val_loss: 0.3917 - val_accuracy: 0.8627

Epoch 3/101875/1875 [==============================] - 9s 5ms/step - loss:
0.3394 -

accuracy: 0.8763 - val_loss: 0.3559 - val_accuracy: 0.8711

Epoch 4/10

1875/1875 [==============================] - 8s 4ms/step - loss: 0.3148 -

accuracy: 0.8857 - val_loss: 0.3555 - val_accuracy: 0.8715

Epoch 5/10

1875/1875 [==============================] - 7s 4ms/step - loss: 0.2961 -

accuracy: 0.8896 - val_loss: 0.3719 - val_accuracy: 0.8682

Epoch 6/10

1875/1875 [==============================] - 16s 8ms/step - loss: 0.2820 -

accuracy: 0.8959 - val_loss: 0.3508 - val_accuracy: 0.8741

Epoch 7/10

1875/1875 [==============================] - 12s 6ms/step - loss: 0.2698 -

accuracy: 0.9003 - val_loss: 0.3496 - val_accuracy: 0.8778

Epoch 8/10

1875/1875 [==============================] - 8s 4ms/step - loss: 0.2596 -

accuracy: 0.9028 - val_loss: 0.3229 - val_accuracy: 0.8854

Epoch 9/10

1875/1875 [==============================] - 9s 5ms/step - loss: 0.2514 -

accuracy: 0.9058 - val_loss: 0.3398 - val_accuracy: 0.8821

Epoch 10/10

1875/1875 [==============================] - 9s 5ms/step - loss: 0.2418 -

accuracy: 0.9094 - val_loss: 0.3262 - val_accuracy: 0.8893

<keras.src.callbacks.History at 0x7a46c8d199c0>
```

```
#predictions of the model
y_pred = model.predict(X_test)
y_pred_classes = y_pred.argmax(axis=1)
313/313 [==============================] - 1s 2ms/step
```

Evaluation Metrics:

```
Accuracy: 0.8893
Precision: 0.8893577804738266
Recall: 0.8893
F1 Score: 0.889103975892847
```

```
#calculation of evaluation metrics
accuracy = accuracy_score(y_test, y_pred_classes)
precision = precision_score(y_test, y_pred_classes, average='weighted')
recall = recall_score(y_test, y_pred_classes, average='weighted')
f1 = f1_score(y_test, y_pred_classes, average='weighted')print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

2.KNN - K-Nearest Neighbors Model

```
KNN Accuracy: 0.8541
KNN Precision: 0.8575414622679564
KNN Recall: 0.8541
KNN F1 Score: 0.8539002124666113
```

```
# Predict the labels for the test set
y_pred_knn = knn.predict(X_test_flat)
# Calculation of evaluation metrics
# Evaluate the performance of the model using accuracy, precision, recall,
and F1 score
accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn, average='weighted')
```

```
recall_knn = recall_score(y_test, y_pred_knn, average='weighted')
f1_knn = f1_score(y_test, y_pred_knn, average='weighted')
# Print the evaluation metrics
# Display the performance of the KNN model
print(f"KNN Accuracy: {accuracy_knn}")
print(f"KNN Precision: {precision_knn}")
print(f"KNN Recall: {recall_knn}")
print(f"KNN F1 Score: {f1_knn}")
```

DISCUSSION

The results show that the ANN model performs better than the KNN model. The ANN model is more capable of learning complex structures, making it more effective in image classification tasks. The data normalization and visualization steps played an important role in improving model performance.

Advantages of ANN:

- **Complex Pattern Learning**: ANN models can learn and represent complex patterns and relationships in the data.
- **High Performance**: In this project, the ANN model achieved higher accuracy and better evaluation metrics compared to the KNN model.
- **Feature Extraction**: ANN models, especially those with multiple layers, can automatically extract important features from raw data.

Disadvantages of ANN:

- **Computationally Intensive**: Training ANN models requires significant computational power and time, especially for large datasets.
- **Hyperparameter Tuning**: ANN models require careful tuning of hyperparameters (e.g., learning rate, number of layers, number of neurons) to achieve optimal performance.

Advantages of KNN:

- **Simplicity**: KNN is easy to implement and understand.
- **No Training Phase**: Unlike many other algorithms, KNN does not have an explicit training phase. It simply stores the training data and performs calculations during the prediction phase.

- **Direct Implementation**: KNN can be applied directly to the dataset without much preprocessing.

Disadvantages of KNN:

- **Distance Metric**: The performance of KNN heavily relies on the choice of distance metric (commonly Euclidean distance).
- **Computationally Intensive**: For large datasets, KNN can be slow as it needs to calculate the distance between the test point and all training points.
- **Curse of Dimensionality**: KNN's performance can degrade with high-dimensional data due to the curse of dimensionality. This is why dimensionality reduction techniques are often applied before using KNN.

Overall, while the KNN model offers simplicity and ease of implementation, the ANN model provides superior performance for complex tasks like image classification. For projects requiring high accuracy and the ability to learn intricate data patterns, ANN models are a better choice despite their higher computational requirements.

7. References:

TensorFlow Documentation

Scikit-learn documentation

keras documentation

Fashion mnist dataset

Wikipedia