Title:

Comparative Analysis of Feedforward, Convolutional, and Recurrent Neural Networks on Dataset Performance

Project Team:

Mohammed Khalid Altufayhi

Anas Mohammed Alsubhi

Faisal Hammad Alomari

Nawaf Abdulrahman Alageel

Albadar Ibrahim Almaymani

Introduction

In recent years, neural networks have become fundamental tools in data analysis, particularly for applications in image recognition, natural language processing, and predictive analytics. Among the varieties of neural networks, the Feedforward Neural Network (FNN), Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN) stand out due to their unique architecture and capabilities in handling different data types and structures. This project aims to investigate and compare these three network models by implementing, training, and testing them on a dataset. This study evaluates each model's accuracy, precision, recall, and F1-score, providing insights into the effectiveness of each approach and offering guidance for future model selection and improvements.

Dataset Description

The dataset used in this study was loaded from a Python library, which is typical for experiments involving large, structured data suitable for image or sequential modeling. The dataset comprises various classes (such as different item types) and includes detailed features that enable a robust classification task. Each data sample is structured in terms of pixels (for image data) or sequences (for sequential data), allowing CNNs to capture spatial relationships and RNNs to capture temporal dependencies. Understanding the nature of the dataset is essential to selecting appropriate preprocessing techniques, hyperparameters, and model architecture, as well as interpreting the model's performance metrics.

Data Exploratory:

Mean: 72.94

Standard Deviation: 90.02

Figure: Sample images from the dataset showing different categories

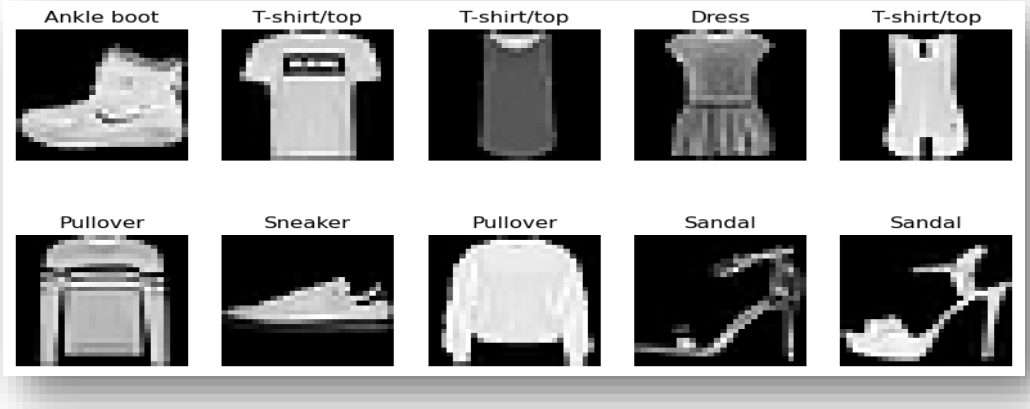


Figure: Average representation of each class in the dataset, showing the characteristic shapes of items

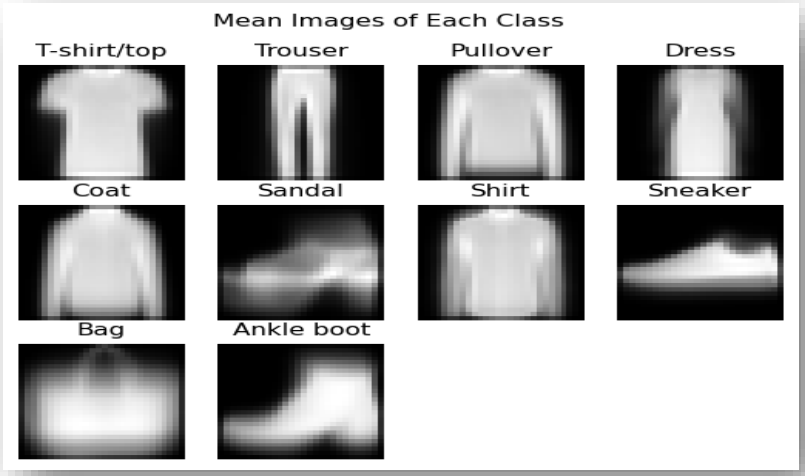
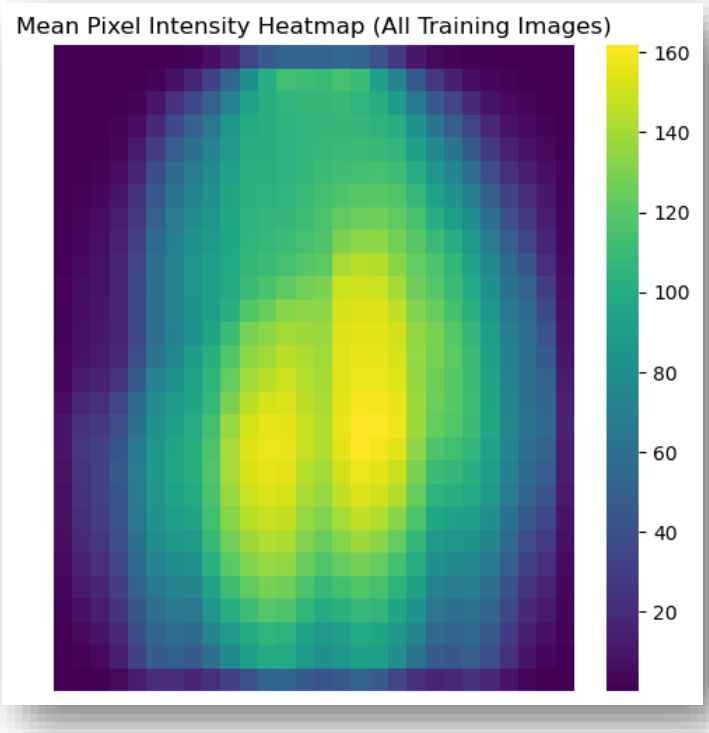


Figure: Heatmap of mean pixel intensities across all training images, illustrating the areas with higher (yellow) and lower (purple) average intensities. This helps in understanding the common patterns and focus areas within the dataset



Data Analysis and Preprocessing

Prior to model implementation, the dataset was analyzed to establish foundational characteristics. Statistical summaries, including means and standard deviations, were calculated to understand the distribution and variance within each feature. Visualizations of data samples provided insights into the variety and complexity present in each class. This exploratory analysis guided the selection of model-specific data preprocessing techniques:

- **Normalization**: Since neural networks benefit from standardized input data, normalization was applied to scale pixel values, enhancing model convergence speed and stability.
- **Label Encoding**: Categorical labels were converted into numerical formats compatible with model architecture, using one-hot encoding for compatibility with classification layers.
- **Train-Test Split**: The dataset was divided into training and test sets, ensuring each model's evaluation could be conducted on unseen data to prevent overfitting.

- ***Each model (FNN, CNN, and RNN) requires different preprocessing due to their unique data handling needs.***
- ***All models used a specific loss function designed for multi-class problems and an optimizer (Adam) that balances speed and accuracy during training.***
- ***During training, each model was evaluated using loss curves to monitor convergence***

FNN Preprocessing: Flattened pixel values to 1D arrays for direct input into dense layers using the code below:

```
# FNN: Flatten the images and normalize pixel values
X_train_fnn = X_train.reshape(X_train.shape[0], -1) / 255.0
X_test_fnn = X_test.reshape(X_test.shape[0], -1) / 255.0
```

FNN Architecture and Hyperparameter: This model is made up of several fully connected layers of neurons (called Dense layers) with ReLU activations, which help the model learn complex patterns. Dropout layers were added to avoid overfitting, where the model learns too much from training data and doesn't perform well on new data. The key factors that were adjusted included the number of neurons in each layer, the learning speed (learning rate), and the number of samples processed at once (batch size).

```
from tensorflow.keras.models import Sequential #A
from tensorflow.keras.layers import Dense, Dropout

# Build the improved FNN model with dropout
fnn_model = Sequential([
    Dense(512, activation='relu', input_shape=(28 * 28,)),
    Dropout(0.3),  # Dropout with 30% rate after the first hidden layer
    Dense(256, activation='relu'),
    Dropout(0.3),  # Dropout with 30% rate after the second hidden layer
    Dense(10, activation='softmax')  # 10 classes for Fashion-MNIST
])

# Compile the model with a slightly reduced learning rate
fnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

from tensorflow.keras.callbacks import EarlyStopping

# Set up early stopping with patience
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

Training Process:

FNN: The loss curve demonstrated a consistent decline with some fluctuations, suggesting variability in the data points. Incorporating dropout regularization helped control overfitting, leading to a more stable loss trend. Ultimately, the model achieved a satisfactory level of accuracy, indicating that the FNN effectively captured the dataset's underlying patterns.

```
fnn_history = fnn_model.fit(
    X_train_fnn, y_train,
    epochs=20,  # Increased epochs to allow early stopping to work effectively
    validation_split=0.2,
    batch_size=64,  # Smaller batch size for more frequent updates
    callbacks=[early_stopping]
)
```

CNN Preprocessing: Maintained spatial dimensions of the data, allowing convolutional filters to capture image patterns using the code below:

```
# CNN: Reshape and normalize the data
X_train_cnn = X_train.reshape(X_train.shape[0], 28, 28, 1) / 255.0
X_test_cnn = X_test.reshape(X_test.shape[0], 28, 28, 1) / 255.0
```

CNN Architecture and Hyperparameter:

This model uses special layers (Convolutional layers) that can identify patterns and features in images. It also includes pooling layers that help reduce the size of the data while keeping important features. Dropout layers were used to improve the model's ability to handle new data. The key adjustments included the size of the filters in convolutional layers and the learning speed.

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten #8
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam


# Build the CNN model
cnn_model = Sequential([
    Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.3),

    Conv2D(128, kernel_size=(5, 5), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.2),

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])

cnn_model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Training Process:

The CNN model learned quickly, taking advantage of its ability to recognize patterns in image data. Although there were small signs of overfitting, these were controlled by using dropout and batch normalization layers. The model's fast learning showed how effective CNNs are at handling image-based tasks.

```
cnn_history = cnn_model.fit(X_train_cnn, y_train, epochs=20, validation_split=0.2, batch_size=128)
```

RNN Preprocessing: Reshaped data to sequences, permitting sequential layer processing to retain information across time steps using the code below:

```
# RNN: Reshape the data to treat each row as a time step
X_train_rnn = X_train.reshape(X_train.shape[0], 28, 28) / 255.0
X_test_rnn = X_test.reshape(X_test.shape[0], 28, 28) / 255.0
```

RNN Architecture and Hyperparameter: This model uses a type of special layer called LSTM (Long Short-Term Memory), which helps it remember information from sequences, making it great for tasks where order matters. The model includes fully connected output layers, and the key settings adjusted were the number of LSTM layers, the learning speed, and batch size.

```
from tensorflow.keras.layers import LSTM #M

# Build the RNN model
rnn_model = Sequential([
    LSTM(64, input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

rnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Training Process:

The RNN model took longer to learn because processing sequences through LSTM layers is more complex. However, the loss eventually leveled off, showing that the model could learn patterns over time. Although RNNs are usually used for tasks with sequential data, this experiment showed that they can still perform reasonably well on structured datasets like images.

```
rnn_history = rnn_model.fit(X_train_rnn, y_train, epochs=20, validation_split=0.2)
```

Evaluation and Results

Upon completing the training phase, each model was evaluated on a separate test set. Key performance metrics, including precision, recall, F1-score, and accuracy, were calculated to provide a comprehensive assessment of each model's classification performance.

Feedforward Neural Network (FNN)

- **Accuracy:** The model reached about 89% accuracy on the Fashion-MNIST test data.
- **Precision:** The model was accurate in most categories, but had a slight drop in classes like "Shirt" and "Pullover," which were more challenging.
- **Recall:** The overall recall was good, but it was lower for classes that looked similar, such as "Shirt" and "T-shirt/top."
- **F1-score:** The F1-score showed a good balance between precision and recall, but there's still a chance to improve for classes that have overlapping features.

Convolutional Neural Network (CNN)

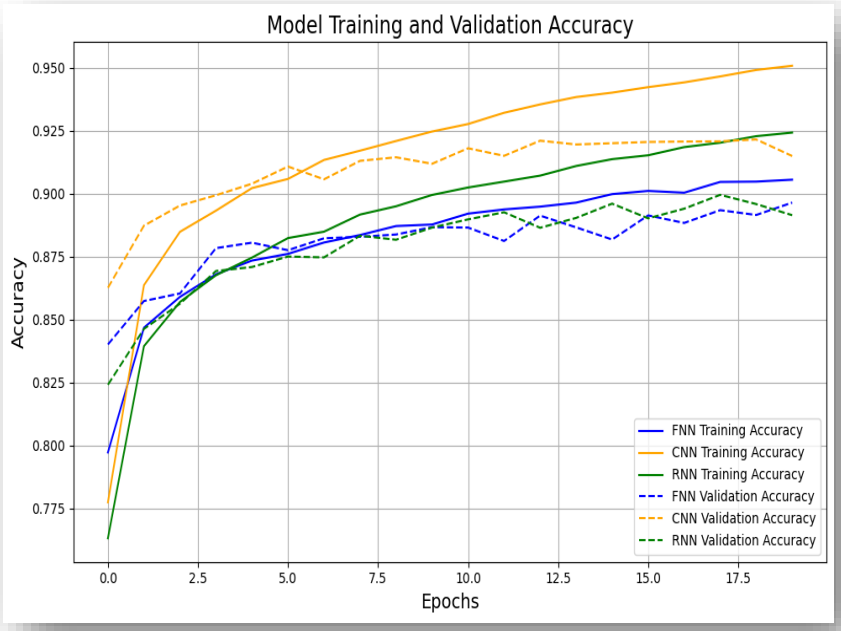
- **Accuracy:** The model correctly predicted 91% of all test items.
- **Precision:** The model was very accurate in predicting most classes. It did best with "Trouser" and "Bag" (both 0.99) and had a lower accuracy for "Shirt" (0.69).
- **Recall:** The model did well in identifying most items in each class. It had the highest recall for "Trouser" and "Sandal" (0.98) but was lower for "T-shirt/top" (0.82).
- **F1-score:** The F1-scores showed a good balance between accuracy and identifying items correctly. It was highest for "Sneaker" and "Ankle boot" (0.97) and lowest for "Shirt" (0.75).

Recurrent Neural Network (RNN)

- **Accuracy:** The model correctly predicted 89% of all items in the test set.
- **Precision:** The model was accurate in predicting most classes, with high precision for "Trouser" and "Sandal" (0.98). The lowest precision was for "Shirt" (0.66).
- **Recall:** The model was good at identifying items in each class, with the highest recall for "Sneaker" and "Ankle boot" (0.97) and the lowest for "T-shirt/top" (0.79).
- **F1-score:** The F1-scores showed a decent balance between accuracy and identifying items correctly, with the highest scores for "Sandal" and "Ankle boot" (0.98) and the lowest for "Shirt" (0.72).

Discussion and Model Comparison

The results showed that CNNs achieved the highest overall performance compared to FNNs and RNNs on this image-based dataset. CNNs demonstrated superior accuracy and metrics due to their ability to efficiently capture spatial hierarchies through convolutional layers. The FNNs performed adequately in identifying general patterns but were limited by their flat architecture, preventing in-depth feature extraction. RNNs, which are primarily suited for sequential data, face challenges in interpreting static image inputs. However, they managed to retain and utilize ordered information within the images, resulting in moderate performance. Ultimately, CNNs outperformed both alternatives in classification accuracy and loss consistency across most classes.



Accuracy Plot: Shows how each model's accuracy improved over time. CNN had the best and most consistent accuracy, while FNN and RNN had more fluctuations.

Loss Plot: Shows the decrease in loss for each model over time. CNN consistently had the lowest training and validation loss, while the RNN had more variability, indicating challenges in maintaining stability.

Key observations include:

- **Model Complexity**: CNNs used the most memory and processing power because of their complex layers, but they gave the best results. RNNs also needed a lot of processing power but didn't show a similar improvement in accuracy for this dataset.
- **Training Stability**: FNN and CNN models showed smoother and more consistent learning curves, while RNNs took longer to learn and showed more ups and downs during training. This is common for models that work with sequences.
- **Generalization:** CNNs performed better on new, unseen data, handling variations in the dataset well. FNNs and RNNs were more affected by changes in the data, indicating less robust generalization.

Conclusion

This study compares three types of neural network models (FNN, CNN, and RNN) using an image dataset. CNN showed the best performance because it uses spatial hierarchies, achieving the highest accuracy. The FNN, while simpler, was still effective at identifying overall patterns in the data. The RNN, though not designed for image data, showed some adaptability and did moderately well.

For future research, using CNNs or combining CNN and RNN layers in hybrid models could improve performance on tasks that need both spatial and sequence-based understanding, like video or image sequences. This project helps in choosing the right model based on the type of data and emphasizes the importance of data preprocessing and fine-tuning model settings.