Enhancing Fashion Image Classification: A Comparative Study of Baseline and Augmented CNN Architectures

1. Overview

In this tutorial, we shall understand how Convolutional Neural Networks (CNNs) can be used to effectively classify clothing image based on Fashion MNIST dataset. In particular, it compares one particular data augmentation baseline CNN model with a baseline CNN model. The intention is to show which of the data enrichment techniques are the best architectural choices that will improve the model generalization and performance, as well as the guidance on reproducibility and exploration.

2. Objectives and Motivation

Objectives

- Demonstrate the implementation of a CNN using TensorFlow/Keras for image classification.
- Compare the performance of a basic CNN model versus one trained with data augmentation.
- Explain the rationale behind each architectural choice and preprocessing step.
- Visualize training and validation metrics to support insights on model performance.
- Educate others on how to apply these techniques for robust fashion image classification.

Motivation

Fashion MNIST is also a modern benchmarking dataset characterized by moderately complex data that poses challenges to the standard CNN. Data augmentation is also integrated to train the model from varied visual variations that will decrease overfitting and improve performance on unseen images. The purpose of this tutorial is to give the practitioners the theory and tools to enable them to use image classification effectively. (Keras/TensorFlow, 2012)

3. Dataset Description

Fashion-MNIST Overview

- **Source:** Provided by TensorFlow/Keras.
- **Composition:**
    - 60,000 grayscale training images (28×28 pixels).
    - 10,000 grayscale test images.
- **Classes:** 10 categories representing various clothing items such as T-shirts, trousers, dresses, sneakers, etc.

Why Use Fashion-MNIST?

- **Moderate Complexity:** Offers a challenging yet manageable dataset for exploring CNN performance. (Keras/TensorFlow, 2012)
- **Balanced Classes:** Ensures equitable representation across all categories.
- **Accessibility:** Easily available within the TensorFlow/Keras ecosystem, facilitating rapid prototyping and experimentation. (Keras/TensorFlow, 2012)

4. Experimental Setup

Environment and Tools

- **Programming Language:** Python 3.10
- **Deep Learning Framework:** TensorFlow 2.1 / Keras
- **Development Platform:** Google Colab (ensuring GPU acceleration and ease of sharing)
- **Visualization:** Matplotlib for plotting training curves and performance metrics.

Data Preprocessing

- **Normalization:** Pixel values are scaled to the range [0, 1] by dividing by 255.0.
- **Reshaping:** Images are expanded to have a single channel, transforming shape from (28, 28) to (28, 28, 1).
- **Label Encoding:** Class labels are converted into one-hot encoded vectors for multi-class classification.

```
1 # Normalize the images
2 train_images = train_images / 255.0
3 test_images = test_images / 255.0
```

5. Model Architectures

Model Comparison is done between two Convolutional Neural Network (CNN) models, one trained on the raw Fashion-MNIST dataset and another one where data augmentation is used to see the effect of augumentation on the classifying performance of the model.

5.1 Baseline CNN Model

The baseline CNN serves as a reference point, using a straightforward architecture without data augmentation. (Srivastava, 2014)

Architecture:

1. **Input Layer:**

- Accepts grayscale images of size **(28, 28, 1)**.

2. **Convolutional Block 1:**
   - **32 filters** with a **5×5 kernel**: Detects edges and textures.
   - **ReLU activation**: Introduces non-linearity.
   - **Batch Normalization:** Stabilizes training and improves convergence.
   - **MaxPooling (2×2)**: reduces spatial dimensions and computational cost.
3. **Convolutional Block 2:**
   - **64 filters** with a **3×3 kernel**: Detects more complex patterns.
   - **ReLU activation** and **Batch Normalization** for stable learning.
   - **MaxPooling (2×2)** for further downsampling.
4. **Flattening:**
   - Converts 2D feature maps into a 1D vector for the dense layer.
5. **Dense Layer:**
   - **256 neurons** with **ReLU activation**: Learns high-level abstract features.
6. **Dropout:**
   - **Rate of 0.4**: Prevents overfitting by randomly deactivating 40% of neurons.
7. **Output Layer:**
   - **10 neurons** with **softmax activation**: Outputs class probabilities for multi-class classification.

5.2 CNN with Data Augmentation

This model builds on the baseline CNN but incorporates a data augmentation pipeline to improve generalization and reduce overfitting.

Data Augmentation Techniques:

- **Rotations:** Up to **15°** → Helps the model handle different orientations.
- **Horizontal/Vertical Shifts:** Up to **15%** of width/height → Improves spatial invariance.
- **Zooming and Shearing:** Expands or distorts the image to teach shape variations.
- **Horizontal Flips:** Teaches the model left-right symmetry.

```python
# Data Augmentation Strategy
augment_data = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.15,
    height_shift_range=0.15,
    zoom_range=0.1,
    shear_range=0.1,
    horizontal_flip=True
)

```

```python
# Train with augmented data
augmented_model = create_cnn_model()
augmented_history = augmented_model.fit(
    augment_data.flow(train_images, train_labels, batch_size=64),
    epochs=12,
    validation_data=(test_images, test_labels)
)
```

Training Strategy:

- Augmented images are generated **in real time** using ImageDataGenerator.
- The model sees slightly altered versions of the same image during each epoch.
- This increases dataset diversity and prevents memorization, encouraging the model to focus on key patterns rather than noise.

Key Differences between the Models:

Aspect	Baseline CNN	Augmented CNN
Training Data	Original dataset only	Augmented with rotated, shifted, zoomed images
Convergence	Faster	Slower due to added complexity
Generalization	Prone to overfitting	Better generalization
Validation Accuracy	Higher initially	Improves gradually with more epochs
Complexity	Moderate	Higher due to augmented data complexity

Why This Approach is Effective

1. The **Baseline Model** sets a performance benchmark under controlled conditions.
2. The **Augmented Model** improves generalization by introducing variability in the training set.
3. Batch normalization and dropout ensure stable training despite the increased complexity from augmentation.

6. Training and Evaluation

Training Details

- **Baseline Model:**
  - Trained for 12 epochs with a batch size of 64.
  - Validation split set to 15% of training data.
- **Augmented Model:**
  - Also trained for 12 epochs with a batch size of 64.
  - Uses an ImageDataGenerator to feed augmented images.
  - Evaluation performed on the separate test set.

Performance Metrics

Both models are evaluated based on:

- **Accuracy:** Proportion of correctly classified images.
- **Loss:** Categorical cross-entropy, indicating model convergence.
- **Training vs. Validation Curves:** Plotted to analyze learning behavior and potential overfitting.

7. Results and Analysis

Baseline Model Performance

- **Training Accuracy:** Improved steadily, reaching over 95% by epoch 12.
- **Validation Accuracy:** Stabilized around 92%, indicating good initial generalization.
- **Training Dynamics:** The loss decreased consistently, though validation loss showed some fluctuation, suggesting minor overfitting tendencies.

Augmented Model Performance

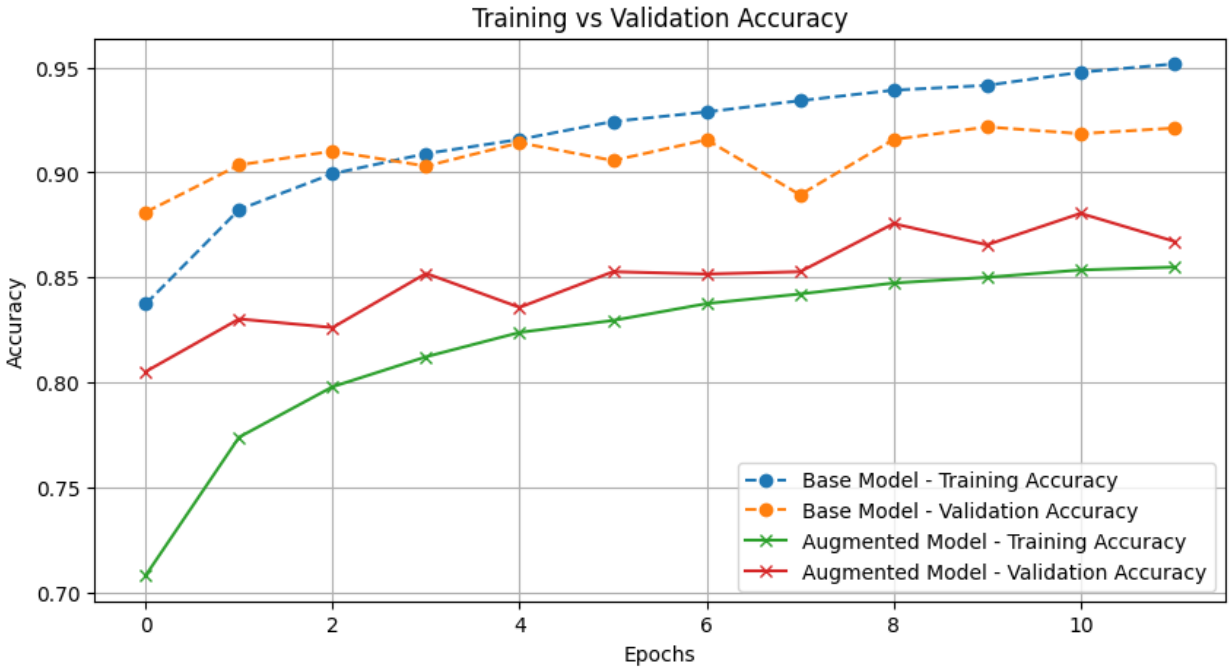
- **Training Accuracy:** Lower initial accuracy (~65%) due to data variability, gradually improving to around 85% by the end of training.
- **Validation Accuracy:** Achieved approximately 87–88% on the test set.
- **Learning Behavior:**
    - The gap between training and validation metrics was narrower compared to the baseline.
    - Augmentation introduced additional complexity, which slowed down convergence but resulted in more robust generalization.

```
938/938 ━━━━━━━━━━━━━━━━━━━━ 31s 28ms/step - accuracy: 0.6473 - loss: 1.0304 - val_accuracy: 0.8051 - val_loss: 0.5040
Epoch 2/12
938/938 ━━━━━━━━━━━━━━━━━━━━ 22s 23ms/step - accuracy: 0.7665 - loss: 0.6241 - val_accuracy: 0.8302 - val_loss: 0.4378
Epoch 3/12
938/938 ━━━━━━━━━━━━━━━━━━━━ 22s 24ms/step - accuracy: 0.7914 - loss: 0.5533 - val_accuracy: 0.8261 - val_loss: 0.4498
Epoch 4/12
938/938 ━━━━━━━━━━━━━━━━━━━━ 22s 24ms/step - accuracy: 0.8065 - loss: 0.5120 - val_accuracy: 0.8519 - val_loss: 0.3960
Epoch 5/12
938/938 ━━━━━━━━━━━━━━━━━━━━ 22s 24ms/step - accuracy: 0.8228 - loss: 0.4775 - val_accuracy: 0.8357 - val_loss: 0.4379
Epoch 6/12
938/938 ━━━━━━━━━━━━━━━━━━━━ 22s 23ms/step - accuracy: 0.8261 - loss: 0.4707 - val_accuracy: 0.8527 - val_loss: 0.3879
Epoch 7/12
938/938 ━━━━━━━━━━━━━━━━━━━━ 21s 22ms/step - accuracy: 0.8354 - loss: 0.4429 - val_accuracy: 0.8516 - val_loss: 0.4159
Epoch 8/12
938/938 ━━━━━━━━━━━━━━━━━━━━ 23s 24ms/step - accuracy: 0.8419 - loss: 0.4253 - val_accuracy: 0.8527 - val_loss: 0.4038
Epoch 9/12
938/938 ━━━━━━━━━━━━━━━━━━━━ 22s 24ms/step - accuracy: 0.8470 - loss: 0.4143 - val_accuracy: 0.8756 - val_loss: 0.3411
Epoch 10/12
938/938 ━━━━━━━━━━━━━━━━━━━━ 22s 24ms/step - accuracy: 0.8483 - loss: 0.4103 - val_accuracy: 0.8655 - val_loss: 0.3687
Epoch 11/12
938/938 ━━━━━━━━━━━━━━━━━━━━ 21s 23ms/step - accuracy: 0.8558 - loss: 0.3897 - val_accuracy: 0.8805 - val_loss: 0.3262
Epoch 12/12
938/938 ━━━━━━━━━━━━━━━━━━━━ 42s 24ms/step - accuracy: 0.8545 - loss: 0.3962 - val_accuracy: 0.8672 - val_loss: 0.3684
```

Comparative Insights

- **Baseline vs. Augmented:**
    - The baseline model achieved higher training accuracy and slightly better validation performance on the held-out split.
    - The augmented model, despite a slower start, demonstrated consistent performance on unseen test data, highlighting the benefits of augmentation in mitigating overfitting on real-world data.

- **Visualizations:**
  - Accuracy curves and loss plots illustrate the trade-off between fast convergence and model robustness.
  - The augmented model's curves suggest improved stability and resistance to overfitting.



8. Discussion

Key Observations

- **Impact of Data Augmentation:**
  - Augmentation introduces variability that forces the model to learn invariant features, thus improving generalization.
  - Although the training accuracy is lower, the augmented model maintains competitive performance on the test set, which is critical for real-world applications.
- **Model Robustness:**
  - Dropout and Batch Normalization complement each other to stabilize training.
  - The combination of these techniques in both models underscores the importance of architectural choices in combating overfitting. (Srivastava, 2014)

Lessons Learned

- **Trade-offs in Model Design:**
  - A simpler baseline can quickly achieve high accuracy but may not generalize well to new data.
  - Incorporating augmentation and regularization techniques can lead to a more resilient model, albeit with a slightly slower training process.
- **Practical Implications:**
  - The approaches demonstrated here are transferable to other image classification tasks where data scarcity or overfitting is a concern.
  - Future work could explore hyperparameter tuning (e.g., adjusting dropout rates or augmentation parameters) to further optimize performance.

9. Conclusion

In this tutorial, we have investigated in details into improving CNN performance on the Fashion MNIST dataset. First, we show that data augmentation is essential for reduction of overfitting and improved generalization and present the comparison to a baseline CNN and its corresponding augmented counterpart. An analysis is performed and discussed along with visualizations of the performance metrics of the various models, from which practitioners are provided with a clear path in building resilient image classification models.

10. References

- Srivastava, N., et al. (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Journal of Machine Learning Research.
- TensorFlow/Keras Documentation. (2025). *Fashion-MNIST Dataset and Model Building Guidelines.*
- Additional resources on data augmentation and CNN architectures from various online machine learning blogs and research papers.

11. Repository and Accessibility

- **GitHub Repository:** A complete version of the code and additional materials are available at https://github.com/shafiq78a/CNN-Model
- **README and LICENSE Files:** Detailed instructions on how to run the code and reuse the materials under an MIT License are provided. https://github.com/shafiq78a/CNN-Model/blob/main/LICENSE.txt

  https://github.com/shafiq78a/CNN-Model/blob/main/README.md

- **Accessibility Considerations:**
    - Visualizations use high-contrast colors and are accompanied by descriptive captions.
    - The document is structured with clear headings and bullet points for easy navigation.
    - Code comments are provided to assist both novice users and those using screen readers.