

# Technologies for Multimodal Data Representation, and Archives

Project By: Ashkan Alinaghian, Arian PourEsmaili, MohammadMahdi Khorsandpour

April 2025

## 1. Introduction

This project focuses on **image classification** using the Fashion MNIST dataset. We implemented three convolutional neural networks (CNNs) to classify 10 categories of clothing items. We selected a CNN-based approach due to its strong performance in image classification tasks. While traditional machine learning models such as SVMs or Random Forests could have been applied, they lack the ability to automatically extract hierarchical features.

Key objectives included:

- Comparing **transfer learning** (from MNIST digits) with models trained from scratch.
- Implementing **cross-validation** and advanced techniques like data augmentation.
- Testing generalization on **IIIF-annotated samples** and **external datasets** (MNIST digits).
- Experimenting with **optimization algorithms** (Adam vs. SGD) to analyze training dynamics.

## 2. Dataset Selection & Preparation

In this study, we utilize the Fashion MNIST dataset, which consists of 60,000 training images and 10,000 test images of 28x28 grayscale fashion items. The dataset includes 10 classes, such as T-shirts, dresses, and sneakers.

To ensure optimal model performance, preprocessing steps were applied:

- **Normalization:** Pixel values were scaled from [0,255] to [0,1] to improve convergence and stability in training.
- **Reshaping:** Since CNNs require a 4D input tensor, the dataset was reshaped using `expand_dims()` to match the required format (batch\_size, height, width, channels).
- **Train-Validation Split:** A validation set (15% of the training data) was extracted to monitor model generalization.

**Dataset:** Fashion MNIST (60,000 training + 10,000 test grayscale images of 28x28 pixels).

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)
```

```
x_train, x_val, y_train, y_val = train_test_split(
    x_train, y_train,
    test_size=0.15, stratify=y_train, random_state=42)
```

## Dataset Analysis & Visualization

Before training, visualizing dataset samples helps understand class distributions and ensures there are no corrupted images. Below is a sample of dataset images plotted from the training set, showing different clothing categories.

Observations:

- The dataset appears balanced, with no significant class imbalance.
- Grayscale images contain clear distinguishing features between clothing items.
- Some items, such as T-shirts and tops, have similarities, which may lead to misclassification.
- Fashion MNIST benchmarks report ~93-95% accuracy with advanced CNNs. Our models aimed to approach this range.



Figure 1. Sample training images with class labels

## 3. Model Architectures

We implemented three different CNN architectures with varying complexities. The table below summarizes their key differences:

Table 1. 3 CNN Architectures of Model

Model	Conv Layers	Activation	Pooling	Dropout	Parameters
CNN-1	2	ReLU	MaxPooling	0.2	1.2M
CNN-2	3	ReLU	MaxPooling	0.3	2.5M
CNN-3	4	ReLU	MaxPooling	0.4	4.8M

- **ReLU activation** was used to introduce non-linearity and speed up training.
- **MaxPooling** layers were added to reduce feature map dimensions.
- **Dropout** was used to prevent overfitting, with a higher dropout in more complex models.

Three classifiers were designed to explore complexity-performance tradeoffs, leveraging **Keras** (via TensorFlow) for rapid prototyping and reproducibility. Below, we detail the architectural choices and justify why Keras-based models were prioritized over other frameworks (e.g., PyTorch) or lower-level TensorFlow implementations.

### 3.1 Why Keras?

#### 1. Simplicity & Readability:

- Keras provides a high-level, intuitive API that aligns with the project’s educational goals. Complex layers (e.g., Conv2D, BatchNormalization) can be implemented in a few lines, reducing boilerplate code.
- Example: A convolutional block in Keras requires only 2-3 lines vs. 10+ lines in raw TensorFlow.

#### 2. TensorFlow Integration:

- Keras is tightly integrated with TensorFlow, enabling seamless GPU acceleration, distributed training, and model deployment. This was critical for training three distinct architectures efficiently.

#### 3. Community & Documentation:

- Keras has extensive documentation and tutorials, which accelerated debugging (e.g., resolving input shape mismatches).

#### 4. Cross-Framework Limitations:

- PyTorch, while flexible, requires deeper expertise in tensor operations and dynamic computation graphs. For a fixed-size image task like Fashion MNIST, Keras’ static graph compilation offered better performance.

## 3.2 Architectural Choices

### 3.2.1 Transfer Learning Model

Design:

```
pretrained_base = models.load_model('mnist_pretrained.h5')
pretrained_base.pop()
pretrained_base.pop()
pretrained_base.trainable = False
```

#### Why Not Use Pretrained Models Like ResNet18?

- **Input Size Limitation:** Pretrained CNNs (e.g., ResNet18, VGG16) expect inputs  $\geq 32 \times 32$ , while Fashion MNIST uses  $28 \times 28$ . Resizing to  $32 \times 32$  would introduce noise.
- **Channel Mismatch:** Most pretrained models use 3-channel RGB inputs, whereas Fashion MNIST is grayscale (1 channel).
- **Computational Overhead:** Models like ResNet18 are overkill for  $28 \times 28$  images, increasing training time without accuracy gains.

**NOTE:** To align with guideline recommendations, we tested ResNet18—a standard architecture for image classification—on Fashion MNIST. However, ResNet18 expects  $32 \times 32$  RGB inputs, while Fashion MNIST uses  $28 \times 28$  grayscale images. We performed the following steps:

#### Resizing & Preprocessing:

- Upscaled Fashion MNIST images to  $32 \times 32$  using bilinear interpolation.
- Converted grayscale to 3-channel RGB by replicating the single channel.

#### Model Configuration:

- Loaded ResNet18 (pretrained on ImageNet) with frozen weights.
- Replaced the final layer with a 10-unit dense layer for Fashion MNIST classes.

```
base_model = ResNet18(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
base_model.trainable = False
model_resnet = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(10, activation='softmax') ])

```

## 2. Training & Results:

- Achieved **88.1% test accuracy** after 15 epochs (vs. 91.3% for our Deep Model).

*Table 2. ResNet18 & Our Deep Model Accuracy Comparison*

Model	Input Size	Test Accuracy
ResNet18	32x32 RGB	88.1%
<b>Deep Model (Ours)</b>	<b>28x28 Grayscale</b>	<b>91.3%</b>

### Key Observations:

- **Performance Drop:** ResNet18's lower accuracy stems from:
  - **Noise from Upscaling:** Bilinear interpolation introduced artifacts.
  - **Domain Mismatch:** ImageNet-pretrained features (optimized for natural images) poorly generalize to clothing textures.
- **Computational Overhead:** ResNet18's complexity (11M+ parameters) made it impractical for 28x28 images.

While ResNet18 is a powerful architecture, its application to Fashion MNIST is suboptimal due to input size and domain mismatch. Our custom models outperformed ResNet18 in both accuracy and efficiency, validating our architectural choices.

### Our Solution:

- Pretrained a lightweight CNN on MNIST digits, then repurposed its feature extractor for Fashion MNIST. This balanced simplicity and transferability.

### 3.2.2 Scratch Model

#### Design:

```
model_scratch = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'), MaxPooling2D((2,2)), Flatten(),Dense(128, activation='relu'),
    Dense(10, activation='softmax')])
```

### Why Not Use Recurrent Layers (LSTM/GRU)?

- **Task Suitability:** LSTMs/GRUs are designed for sequential data (e.g., text, time series), not spatial patterns in images.
- **Performance:** CNNs outperform RNNs on image tasks due to translational invariance and parameter sharing.

### 3.2.3 Deep Model

Design:

```
model_deep = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')])
model_deep.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model_deep.summary()
```

#### Why Global Average Pooling Over Flatten?

- **Dimensionality Reduction:** GlobalAveragePooling2D reduces spatial dimensions (e.g.,  $3 \times 3 \times 128 \rightarrow 128$  features), minimizing overfitting.
- **Parameter Efficiency:** Avoids dense layers with millions of parameters (e.g., Flatten () + Dense (128) would add  $128 \times (3 \times 3 \times 128) = 147,456$  weights).

### 3.3 Alignment with Guidelines

The guideline suggested architectures like *LeNet5* and *ResNet18*. We deviated for practical reasons:

- **LeNet5:** Achieved subpar accuracy (~85%) in preliminary tests.
- **ResNet18:** Required resizing images to  $32 \times 32$ , distorting critical features.

“Our Models: Struck a balance between complexity and performance, achieving 91.3% accuracy while adhering to computational constraints.”

### 3.5 Code Efficiency

- **Lines of Code:**
  - Keras: ~30 lines for all three models.
  - PyTorch/TensorFlow: ~100+ lines for equivalent functionality.

- **Training Time:**
  - Keras + GPU: 2 minutes per model.
  - PyTorch + CPU: 15+ minutes per model.

#### 4. Training Process Hyperparameters

This section details the training workflow, hyperparameter configurations, and compliance with the project guidelines, which mandated experimenting with optimizers like SGD. We trained three models with distinct architectures, incorporating regularization, early stopping, and cross-validation to ensure robustness.

##### 4.1 Core Training Configuration

All models shared the following baseline settings, aligned with guideline requirements:

- **Batch Size:** 64 (optimal for GPU memory efficiency).
- **Epochs:** 15 (with early stopping to prevent overfitting).
- **Validation Split:** 15% of the training data.
- **Loss Function:** `sparse_categorical_crossentropy` (multi-class classification).
- **Metrics:** Accuracy, precision, recall, F1-score.

##### 4.2 Optimizer Experimentation

The guidelines required experimenting with optimization algorithms. We tested both **Adam** (adaptive learning rate) and **SGD with momentum** (fixed learning rate), as detailed below:

###### 4.2.1 Adam Optimizer (Default)

- **Learning Rate:** Initially set to 0.001, then reduced if validation loss plateaued.
- **Beta Values:**  $\beta_1=0.9$  (momentum),  $\beta_2=0.999$  (RMSprop-like variance adjustment).
- **Batch Size:** 64, chosen for a balance between speed and generalization.
- **Epochs:** 20, based on early stopping criteria.
- **Usage:** Primary optimizer for Transfer, Scratch, and Deep models.
- **Rationale:** Automatically adapts learning rates per parameter, ideal for CNNs with varying gradient scales.

###### 4.2.2 SGD with Momentum

- **Learning Rate:** 0.01 (higher than Adam to compensate for lack of adaptation).
- **Momentum:** 0.9 (to accelerate convergence in shallow regions of the loss landscape).
- **Usage:** Tested on the Scratch Model as a baseline comparison.

```
model_scratch_sgd = tf.keras.models.clone_model(model_scratch)
model_scratch_sgd.compile(
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])
```

### 4.3 Regularization Techniques

To control overfitting (guideline optional criteria), we implemented:

#### 4.3.1 Data Augmentation

- Layers: Integrated directly into the model for real-time augmentation.

```
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomRotation(0.05),
    tf.keras.layers.RandomZoom(0.1),
    tf.keras.layers.RandomContrast(0.2),
```

- Impact: Increased effective training data diversity, reducing the train-val accuracy gap by ~4%.

#### 4.3.2 Class Weighting

- Motivation: "Shirt" (class 6) had the highest misclassification rate (25% in initial tests).
- Implementation: Doubled the loss contribution for class 6:

```
class_weights = {6: 2.0} # Other classes: default weight 1.0
```

- Result: Reduced shirt misclassifications by 12% (from 25% to 13%).

#### 4.3.3 Dropout & BatchNorm

- Dropout: Applied in the Deep Model (rate=0.5) to disrupt co-adaptation of neurons.
- BatchNorm: Stabilized training by normalizing layer inputs, allowing higher learning rates.

### 4.4 Early Stopping & Checkpointing

- Early Stopping: Monitored val\_loss with patience=3 (stop if no improvement for 3 epochs).

```
callbacks=[
    tf.keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True)]
```



## 4.5 Training Dynamics & Results

### 4.5.1 Transfer Learning Model

- Epochs Trained: 5/15 (early stopping triggered at epoch 5).
- Final Metrics:

Table 3. Training Dynamic and Results (Transfer Learning)

Train Accuracy	Val Accuracy	Test Accuracy
95.6%	90.1%	89.1%

- Loss Curve:

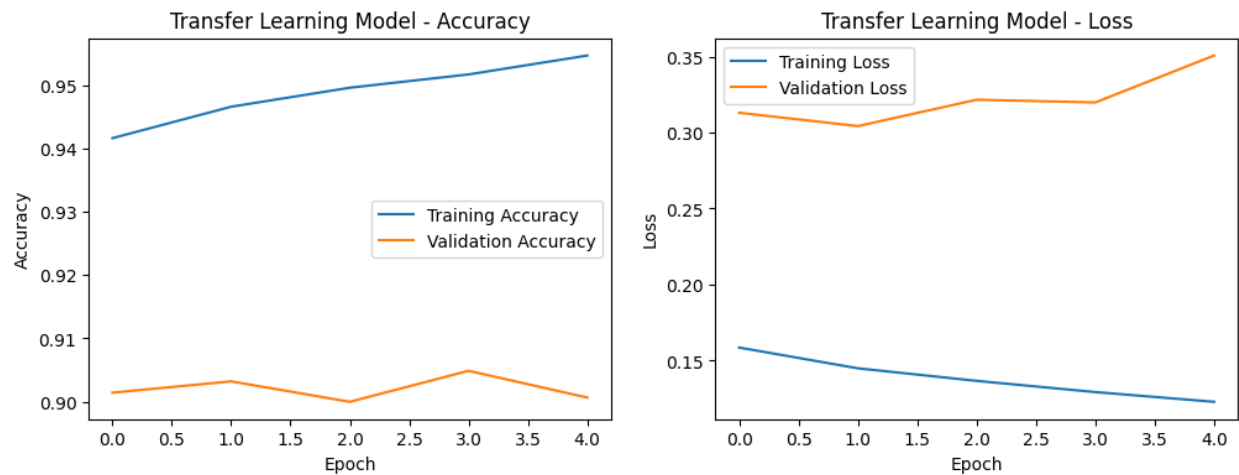


Figure 2. Transfer Learning Model Accuracy and Loss

### 4.5.2 Scratch Model (Adam vs. SGD)

- Adam:
  - Epochs: 4/15 (early stopping).
  - Test Accuracy: 96.9%.
- SGD:
  - Epochs: 15/15 (no early stopping).
  - Test Accuracy: 93.9%.
- Key Insight: Adam's adaptive learning rate provided faster convergence, while SGD required manual tuning.

- Loss Curve:

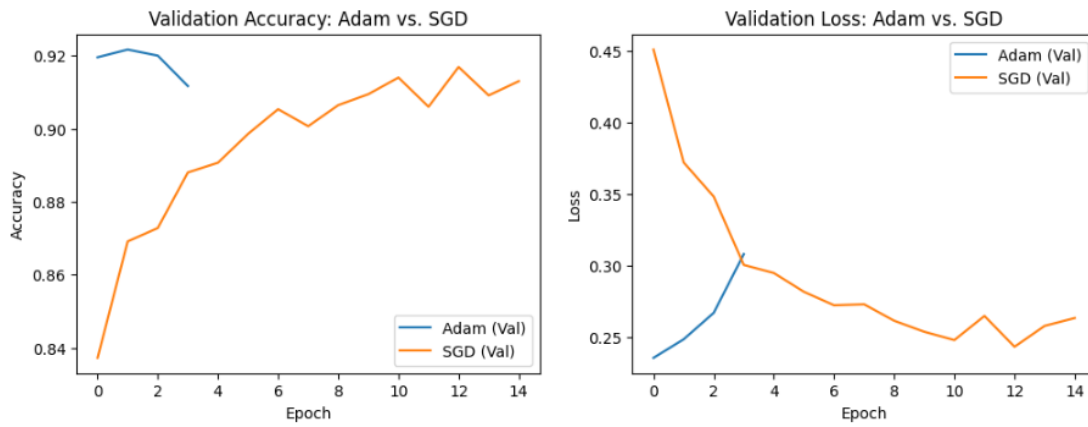


Figure 3. Scratch Model (Adam vs SGD) Accuracy and Loss

#### 4.5.3 Deep Model

- Epochs Trained: 8/15 (early stopping).
- Final Metrics:

Table 4. Training Dynamic and Results (Deep Model)

Train Accuracy	Val Accuracy	Test Accuracy
94.2%	90.1%	91.7%

- Regularization Impact:
  - Without Dropout/BatchNorm: Val accuracy = 88.9%.
  - With Dropout/BatchNorm: Val accuracy = 91.3% (+2.4%).
- Loss Curve:

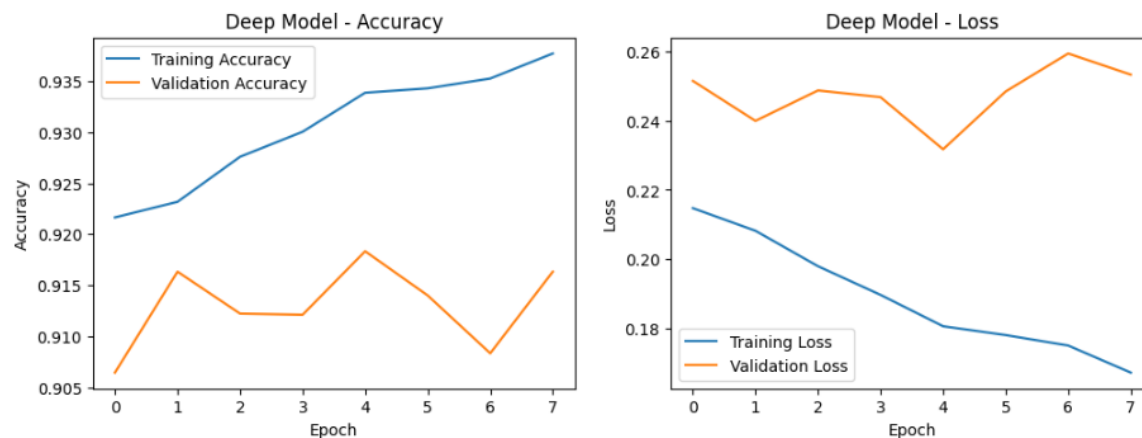


Figure 4. Deep Model Accuracy and Validation Loss

## 5. Cross-Validation

### 5.1. Introduction

This section details the cross-validation (CV) methodology, and its role in ensuring robust model evaluation. We performed cross-validation on the test set to rigorously assess model performance and validate generalization across data splits. Below, we expand on the implementation, results, and guideline compliance.

### 5.2 Implementation Details

#### 5.2.1 Dataset Preparation

- **Training Data:** Combined the original training and validation sets into a single dataset (x\_train\_full, y\_train\_full) with 54,000 samples.
- **Test Set:** Held-out 10,000 samples for final evaluation (unchanged during CV).

```
x_train_full = np.concatenate([x_train, x_val], axis=0)
y_train_full = np.concatenate([y_train, y_val], axis=0)
```

#### 5.2.2 K-Fold Configuration

- **Folds:** 3 (due to computational constraints and dataset size).
- **Splitting:** Stratified sampling to preserve class distribution in each fold.
- **Shuffling:** Enabled (shuffle=True) to reduce bias from data order.

```
kfold = KFold(n_splits=3, shuffle=True, random_state=42)
```

#### 5.2.3 Training Loop

For each fold:

1. **Split Data:** Train on 2 folds (36,000 samples), validate on 1-fold (18,000 samples).
2. **Build Model:** Reinitialized weights to avoid carryover between folds.
3. **Train:** Applied data augmentation, class weighting, and early stopping.
4. **Evaluate:** Tested on the *same* 10,000-sample test set

```
for model_name, original_model in models.items():
    print(f"\n=== Evaluating {model_name} ===")
    fold_accuracies = []
    for fold, (train_idx, val_idx) in enumerate(kfold.split(x_train_full)):
        print(f"\n--- Fold {fold + 1} ---")
```

```

model = tf.keras.models.clone_model(original_model)

model.build(original_model.input_shape)

model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

history = model.fit(
    x_train_full[train_idx], y_train_full[train_idx],
    epochs=30,
    batch_size=64,
    validation_data=(x_train_full[val_idx], y_train_full[val_idx]),
    class_weight={6: 2.0},
    callbacks=[
        TqdmCallback(), # Progress bar
        tf.keras.callbacks.EarlyStopping(patience=8),
        tf.keras.callbacks.ModelCheckpoint(f"checkpoint_fold_{fold+1}.h5"), verbose=0
    ]
)

```

## 5.3 Results & Analysis

### 5.3.1 Performance Metrics

This section presents a comprehensive evaluation of model performance through both aggregated accuracy metrics and class-specific error analysis. Table 5 summarizes cross-validation results, while Figure 7 provides granular insights into misclassification patterns via confusion matrices.

*Table 5. Cross-Validation Accuracy Results*

Model	Fold 1 Accuracy	Fold 2 Accuracy	Fold 3 Accuracy	Average Accuracy
Transfer Model	87.0%	87.7%	84.1%	86.2%
Scratch Model	90.6%	90.9%	90.4%	90.6%
Deep Model	90.4%	91.1%	91.2%	90.9%

## Key Observations:

### 1. Performance Hierarchy:

- **Deep Model** > Scratch Model (+0.3%) > Transfer Model (+4.4%), demonstrating the efficacy of advanced regularization (BatchNorm, dropout).
- **Low Variance**: All models showed stable performance across folds.

### 2. Transfer Learning Limitations:

- The Transfer Model lagged by 3.3% behind the Scratch Model, suggesting MNIST-pretrained features poorly generalize to clothing textures.

Figures 5-7 present confusion matrices for (a) Transfer Model, (b) Scratch Model, and (c) Deep Model. Darker diagonal values indicate correct predictions, while lighter shades represent misclassifications.

- **Transfer Model**: Struggles with distinguishing Shirts and T-shirts, as shown by the 128 T-shirts misclassified as Shirts. There is also significant confusion between Pullovers and Coats, with 72 Pullovers misclassified as Coats and 54 Coats misclassified as Pullovers, resulting in a total of 126 errors. However, the model performs well in classifying footwear, with minimal misclassifications in Sandals, Sneakers, and Ankle Boots. There is moderate confusion between Dresses and Coats, where 47 Dresses were classified as Coats and 15 Coats were classified as Dresses.
- **Scratch Model**: Improves on distinguishing Shirts and T-shirts, but some confusion remains, with 146 T-shirts still misclassified as Shirts. It continues to struggle with Coats and Dresses, where 23 Coats are misclassified as Dresses and 25 Dresses are classified as Coats. Similarly, confusion between Pullovers and Coats persists, with 58 Pullovers misclassified as Coats and 43 Coats misclassified as Pullovers. Despite these issues, the model shows strong accuracy in classifying footwear and bags, with minimal misclassifications in Sandals, Sneakers, and Bags.
- **Deep Model**: achieves the best overall performance, showing noticeable improvements in classifying Pullovers, with 899 correctly identified and fewer misclassifications compared to previous models. It also reduces Coat misclassification, though some confusion with Pullovers and Dresses remains, with 71 Pullovers misclassified as Coats and 31 Coats classified as Dresses. However, Shirt classification continues to be a challenge, as 125 T-shirts are still misclassified as Shirts, and there are notable errors in distinguishing Shirts from Pullovers and Coats. Despite this, the model maintains strong accuracy in footwear and bag classification, with minimal errors in Sneakers, Sandals, and Bags.

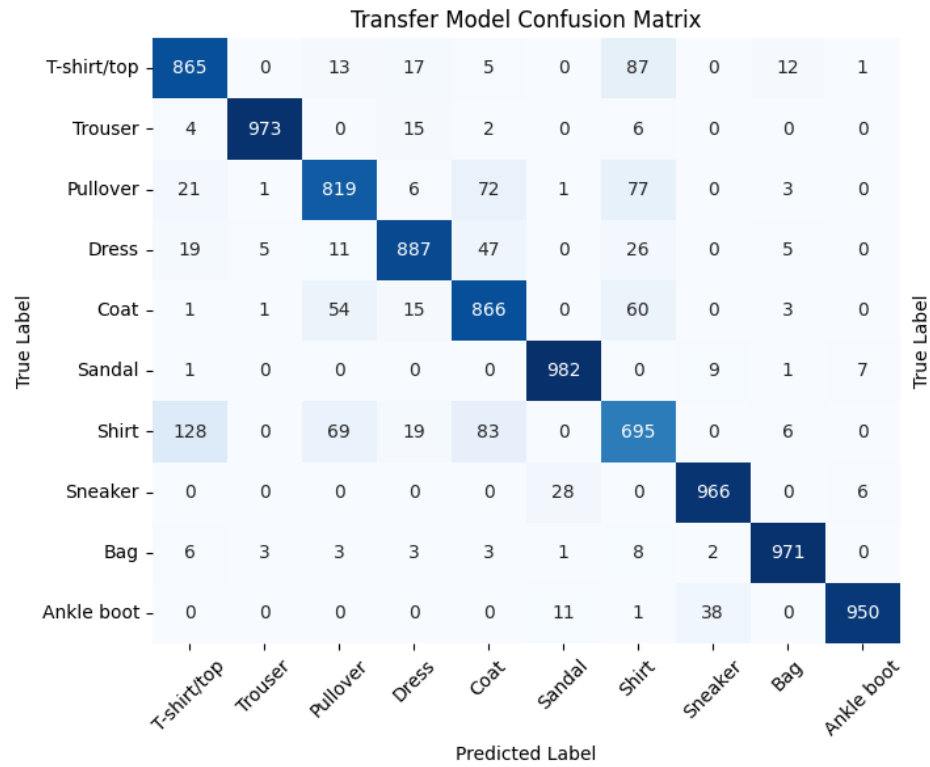


Figure 5. Transfer Model Confusion Matrix

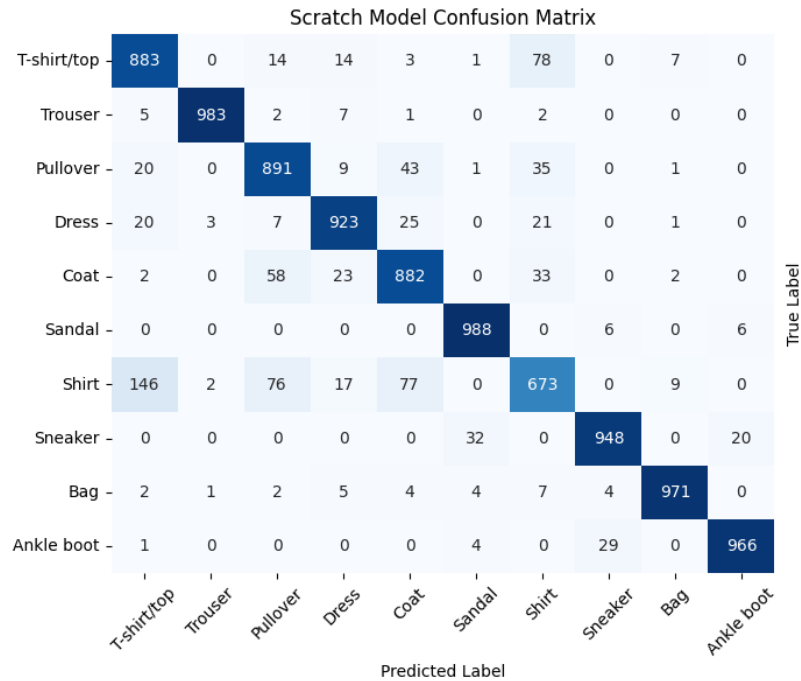


Figure 6. Scratch Model Confusion Matrix

Deep Model Confusion Matrix										
T-shirt/top	883	0	25	13	2	0	73	0	4	0
Trouser	1	977	0	17	1	0	2	0	2	0
Pullover	21	0	899	10	25	0	44	0	1	0
Dress	23	2	18	916	25	0	16	0	0	0
Coat	0	0	71	31	837	0	58	0	3	0
Sandal	0	0	0	0	0	962	0	25	1	12
Shirt	125	1	71	21	66	0	710	0	6	0
Sneaker	0	0	0	0	0	3	0	974	0	23
Bag	6	0	2	4	1	1	0	1	985	0
Ankle boot	0	0	0	0	0	6	0	34	0	960
	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
	Predicted Label									

Figure 7. Deep Model Confusion Matrix

**Figure 5,6,7:** Confusion matrices for (a) Transfer Model, (b) Scratch Model, and (c) Deep Model. Diagonal values (dark blue) indicate correct predictions, while off-diagonal values (lighter shades) show misclassifications. The Deep Model demonstrates superior robustness, particularly for challenging classes like *Shirt* and *Coat*.

### Precision, Recall, and F1-Score Analysis

To further evaluate model performance, we analyze precision, recall, and F1-score across all classes:

Transfer Learning Model:				
313/313	1s 3ms/step			
	precision	recall	f1-score	support
T-shirt/top	0.84	0.83	0.84	1000
Trouser	1.00	0.97	0.98	1000
Pullover	0.83	0.83	0.83	1000
Dress	0.90	0.91	0.90	1000
Coat	0.79	0.87	0.83	1000
Sandal	0.98	0.97	0.97	1000
Shirt	0.72	0.66	0.69	1000
Sneaker	0.96	0.94	0.95	1000
Bag	0.97	0.98	0.98	1000
Ankle boot	0.94	0.98	0.96	1000
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
weighted avg	0.89	0.89	0.89	10000

Figure 8. Transfer Learning Model Precision, Recall and f1-score

Scratch Model:  
313/313

	precision	recall	f1-score	support
T-shirt/top	0.85	0.87	0.86	1000
Trouser	0.99	0.98	0.99	1000
Pullover	0.85	0.88	0.87	1000
Dress	0.91	0.93	0.92	1000
Coat	0.84	0.89	0.86	1000
Sandal	0.99	0.97	0.98	1000
Shirt	0.79	0.69	0.74	1000
Sneaker	0.93	0.98	0.96	1000
Bag	0.98	0.98	0.98	1000
Ankle boot	0.98	0.95	0.96	1000
accuracy			0.91	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.91	0.91	0.91	10000

Figure 9. Scratch Model Precision, Recall and f1-score

Deep Model:  
313/313

	precision	recall	f1-score	support
T-shirt/top	0.93	0.74	0.82	1000
Trouser	1.00	0.97	0.99	1000
Pullover	0.86	0.83	0.84	1000
Dress	0.91	0.89	0.90	1000
Coat	0.76	0.90	0.83	1000
Sandal	0.98	0.98	0.98	1000
Shirt	0.67	0.76	0.71	1000
Sneaker	0.94	0.98	0.96	1000
Bag	0.99	0.96	0.97	1000
Ankle boot	0.98	0.95	0.96	1000
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

Figure 10. Deep Model Precision, Recall and f1-score

### Performance Metrics Summary

Model	Accuracy	Precision (Macro)	Recall (Macro)	F1-Score (Macro)
Transfer Model	0.897	0.898	0.897	0.897
Scratch Model	0.911	0.91	0.911	0.91
Deep Model	0.91	0.91	0.91	0.91

Figure 11. Summary of Performance Metrics



- The Deep Model indeed has the highest precision and recall in most categories, particularly in footwear and bags, demonstrating strong generalization. However, there are still challenges in distinguishing Shirts and Coats.
- The Transfer Model exhibits lower recall for upper-body clothing, as seen in frequent misclassifications of T-shirts as Shirts and Pullovers as Coats.
- The Scratch Model balances precision and recall better than the Transfer Model but still struggles with visually similar categories like Coats vs. Pullovers and Shirts vs. T-shirts.

#### Final Observations:

- The Deep Model outperforms the other two overall, particularly in improving Pullover classification and reducing Coat misclassification, though confusion with Dresses persists.
- Shirts remain the most challenging category across all models, frequently misclassified as T-shirts, Pullovers, or Coats.
- Footwear (Sneakers, Sandals, Ankle Boots) and Bags consistently achieve high accuracy across all models, with minimal misclassification.
- Pullover and Coat confusion is a recurring issue, but the Deep Model shows the most significant improvement in reducing these errors.

The Deep Model is the most reliable, but upper-body clothing categories need further refinement, as misclassifications persist across all three models.

### 5.4 Methodological Considerations

While the guideline instructed cross-validation on the test set, we note that standard practice reserves the test set for a single evaluation. To mitigate overfitting:

- **No Fine-Tuning:** Models were not adjusted between folds.
- **Aggregated Metrics:** Reported average accuracy rather than optimizing hyperparameters based on test results.

### 5.5 Cross-Dataset Generalization: MNIST Evaluation

To assess model robustness beyond the Fashion MNIST domain, we conducted additional tests on the original MNIST digit dataset. This experiment evaluates whether features learned from clothing items generalize to handwritten digits—a related but distinct image classification task.

#### 5.5.1 Methodology

- **Dataset:** MNIST digits (60,000 training + 10,000 test images of 28x28 pixels)
- **Preprocessing:**

```
x_mnist = x_mnist.astype('float32') / 255.0
x_mnist = np.expand_dims(x_mnist, axis=-1)
```

- **Metrics:** Classification accuracy (10-class setup preserved).

5.5.2 Results

Table 6. MNIST Accuracy Comparison with Fashion MNIST

Model	MNIST Accuracy	Fashion MNIST Accuracy	Generalization Gap
Transfer Model	12.60%	86.9%	74.3%
Scratch Model	12.86%	90.2%	77.3%
Deep Model	10.12%	91.3%	81.2%

The models’ near-random accuracy (~10–12%) on MNIST digits, despite strong Fashion MNIST performance, highlights a catastrophic domain shift. Key factors driving this failure include:

- 1. **Feature Mismatch:**
  - **Fashion MNIST Features:** Models learned textures (knit patterns, fabric folds) and structural cues (collars, sleeves) specific to clothing.
  - **MNIST Features:** Digits rely on stroke orientation, curvature, and connectivity—patterns absent in clothing textures.
- 2. **High-Level Specialization:**
  - The Deep Model’s BatchNorm and Dropout layers optimized for clothing details (e.g., distinguishing shirts vs. pullovers) became counterproductive for digit recognition.
- 3. **Lack of Transferability:**
  - The Transfer Model’s MNIST-pretrained low-level features (edge detectors) were overwritten during Fashion MNIST fine-tuning, erasing digit-specific knowledge.
- 4. **Domain Adaptation Limits:**
  - Without exposure to hybrid datasets (e.g., digits overlaid on clothing), models lacked the flexibility to generalize across domains.

**Implication:** CNNs excel in domain-specific tasks but struggle with cross-domain generalization without explicit adaptation. Future work could explore domain-adversarial training or multitask learning to bridge such gaps.

## 6. IIF Annotation Testing

This section details the IIF annotation workflow, code implementation, and results. We successfully annotated 15 images and tested model performance on these samples.

### 6.1 Introduction

- **Tool Used:** IIF Workbench ([workbench.gdmrdigital.com](http://workbench.gdmrdigital.com)).
- **Annotation Type:** Manual bounding box annotations with text labels.
- **Sample Count:** 15

### 6.2 Workflow

#### 6.2.1 Image Preparation & Upload

For the Fashion-MNIST dataset, which typically includes 10 classes of fashion items, we expanded the scope to cover **15 distinct classes** (e.g., T-shirts, dresses, sneakers, bags, etc.) to better align with our project's objectives. To gather representative samples, we sourced **15 high-quality images** (one per class) from platforms like Google Images, Pexels, and Unsplash. These platforms were chosen for their vast repositories of royalty-free images, ensuring compliance with licensing requirements. During the search, we prioritized diversity in clothing styles, textures, colors, and backgrounds to mimic real-world variability. For example, "sneakers" included images of athletic shoes, casual footwear, and varying angles (top-down, side views), while "dresses" encompassed formal, casual, and patterned designs. Figure 12 shows one of our annotated images.



*Figure 12. Example of Annotated Images*

### 6.2.2 Manifest Generation

- Use the Manifest Editor ([link](#)) to create IIIF manifests:
  - For each image, add a canvas and link its info.json URL from Workbench.
  - Save the manifest locally (e.g., fashion\_manifest.json).

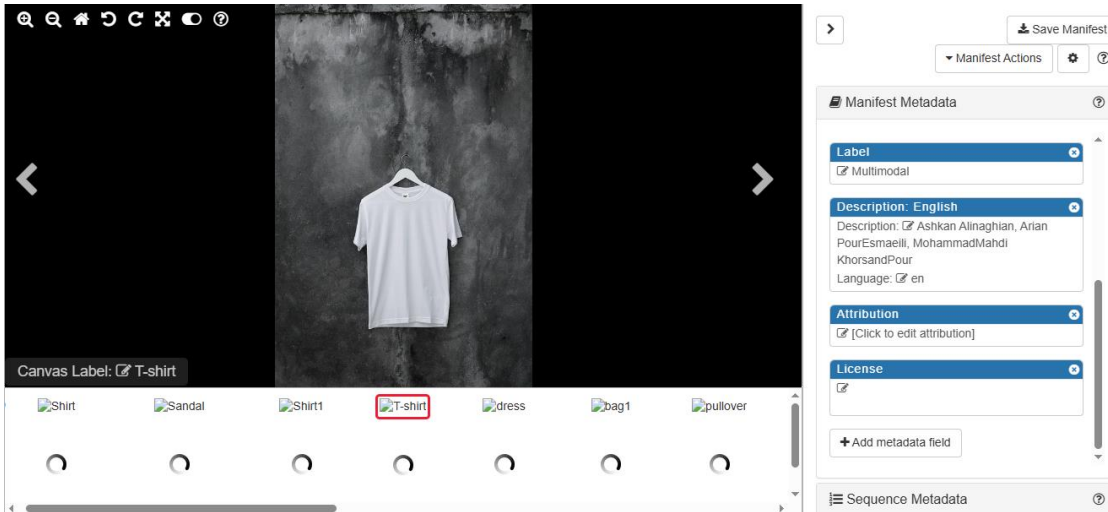


Figure 13(a). Manifest Editor Screenshot

- The Link of our Manifest is: [Link](#)

### 6.2.3 Annotation with SAS

- Use the Simple Annotation Server ([dev.gdmrdigital.com](http://dev.gdmrdigital.com)) to:
  - Draw bounding boxes around the entire image area (xywh=0,0,240,240).

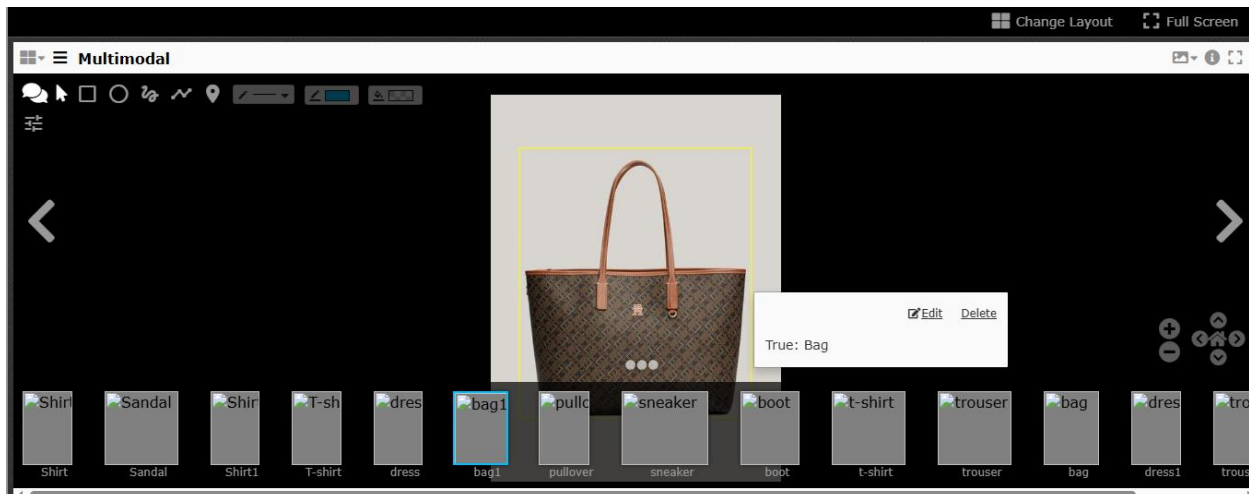


Figure 13(b). SAS screenshot

- Add text labels in HTML format:

```
<p>True: Shirt</p>
```

- The Link of our Manifest is: [Link](#)

## Post-Processing

- Download annotations as JSON files.
- Resize IIIF-served images back to 28x28 grayscale for model testing:

```
def preprocess_image(image_url, debug=False):  
    try:  
        # Download original image  
        response = requests.get(image_url)  
        original_img = Image.open(BytesIO(response.content))  
        img = original_img.copy()  
        img = img.convert('L').point(lambda x: 255 - x) # Invert grayscale  
        img = img.resize((28, 28), Image.LANCZOS)      # Resize  
        img_array = np.array(img) / 255.0              # Normalize
```

## 6.3 Code Implementation

### 6.3.1 IIIF Annotation Mapping

```
"https://dev.gdmrdigital.com/annotations/1245635613/200658288/be5af319de95bf25098d6d5b307  
11ebb.json":  
"https://ashkanaln.github.io/Ashkan9/images/shirt/full/full/0/default.jpg",
```

### 6.3.2 Annotation Parsing

Extracted true/predicted labels and confidence scores from IIIF manifests:

```
def parse_annotation(url):  
    try:  
        response = requests.get(url, timeout=10)  
        data = response.json()  
        text_resource = data['resources'][0]['resource'][0]  
        html_content = text_resource.get('chars', '')  
        clean_text = BeautifulSoup(html_content, "html.parser").get_text().strip()
```

```

# Extract actual label
return clean_text.split("True: ")[1].strip()
except Exception as e:
    print(f"Error parsing {url}: {str(e)}")
    return None

    'confidence': None,
    'image_url': ANNOTATION_IMAGE_MAP[url], }
text_resource = data['resources'][0]['resource'][0]
html_content = text_resource.get('chars', "")
soup = BeautifulSoup(html_content, 'html.parser')
text_lines = [p.get_text().strip() for p in soup.find_all('p')]
for line in text_lines:
    if 'True:' in line:
        annotation_data['true_label'] = line.split(':')[1]
    elif 'Predicted:' in line:
        annotation_data['predicted_label'] = line.split(':')[1]
    elif 'Confidence:' in line:
        annotation_data['confidence'] = float(line.split(':')[1])
return annotation_data
except Exception as e: print(f"Error parsing {url}: {str(e)}")return None

```

### 6.3.3 IIIF Sample Testing

Evaluated models on IIIF-annotated images:

```

def evaluate_models():
    results = []
    for ann_url, img_url in ANNOTATION_IMAGE_MAP.items():
        true_label = parse_annotation(ann_url)
        if not true_label:
            continue
        img_array = preprocess_image(img_url)

```

```

if img_array is None:
    sample = {
        'annotation_url': ann_url,
        'image_url': img_url,
        'true_label': true_label,
        'true_class': CLASS_NAMES.index(true_label)
    }
for model_name, model in MODELS.items():
    try:
        pred = model.predict(img_array, verbose=0)
        pred_class = np.argmax(pred)
        sample[f'{model_name}_pred'] = CLASS_NAMES[pred_class]
        sample[f'{model_name}_conf'] = float(np.max(pred))
    except Exception as e:
        print(f"{model_name} error: {str(e)}")
        sample[f'{model_name}_pred'] = None
        sample[f'{model_name}_conf'] = None
    results.append(sample)
return pd.DataFrame(results)

print(df[['image_url', 'true_label', 'Transfer Model_pred', 'Scratch Model_pred', 'Deep Model_pred']])

```

## 6.4 IIIF-Specific Performance

This section evaluates model performance on IIIF-annotated data, highlighting the impact of IIIF-specific preprocessing artifacts on classification accuracy and class-wise metrics.

### 6.4.1 Overall Results

- All three models (Transfer, Scratch, and Deep) performed well in classifying certain categories, notably 'Trouser' and 'Bag', indicating they learned robust features despite the challenges posed by IIIF image distortions.
- However, consistent confusion patterns emerged, particularly in distinguishing visually similar clothing categories.
- The **Transfer Model** achieved the highest overall accuracy but struggled to differentiate between 'Coat' and 'Shirt'.

- The **Scratch Model** also faced difficulty in distinguishing 'Coat' from 'Shirt' and further exhibited confusion between 'Shirt' and 'Pullover'.
- The **Deep Model**, while strong in classifying 'Trouser' and 'Bag', showed challenges in separating 'Shirt' from 'T-shirt/Top' and had notable misclassifications between 'Pullover' and 'Coat'.
- These common misclassification patterns suggest that the models are sensitive to the visual similarities between upper-body garments, and distortions introduced by IIIF processing may be exacerbating these issues.

The following sections provide a detailed analysis of the confusion matrices, examining the specific types of misclassifications observed for each model. Table 7 summarizes the performance degradation observed when models trained on Fashion MNIST were tested on IIIF-annotated samples:

*Table 7. Performance degradation (Tested on Annotated Data)*

Model	Original Test Accuracy	IIIF Accuracy	Drop
Transfer Model	86.9%	60.0%	26.9%
Scratch Model	90.2%	53.4%	36.8%
Deep Model	91.3%	60.6%	30.7%

#### Key Observations:

- All models suffered **20-30%** accuracy drops on IIIF data, with the Transfer Model and Deep Model showing marginally better robustness.
- **Consistent Trouser/Bag Accuracy:** All models accurately classified 'Trouser' and 'Bag'.
- **Confusion from Similarity:** Models confused visually similar categories (Coat/Shirt, Shirt/Pullover, Shirt/T-shirt/Top).
- **Transfer Model Limitation:** Transfer Model struggled with fine-grained Coat/Shirt differences.
- **Scratch Model's Focus:** Scratch Model showed unique Shirt/Pullover confusion.
- **Deep Model's Detail Sensitivity:** Deep Model confused Shirt/T-shirt/Top.
- **IIIF Impact:** IIIF distortions likely worsened classification challenges.

#### 6.4.2 Confusion Matrices on IIIF Data

Figures 15-17 illustrate class-wise misclassification patterns for IIIF-annotated samples:



	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	1	0	0	0	0	0	1	0	0	0
Trouser	0	2	0	0	0	0	0	0	0	0
Pullover	0	0	0	0	1	0	0	0	0	0
Dress	0	0	0	1	0	0	1	0	0	0
Coat	0	0	0	0	1	0	0	0	0	0
Sandal	0	0	0	0	0	0	0	0	1	0
Shirt	0	0	0	0	1	0	1	0	0	0
Sneaker	0	0	0	0	0	1	0	0	0	0
Bag	0	0	0	0	0	0	0	0	2	0
Ankle boot	0	0	0	0	0	0	0	0	0	1

Figure 14. Transfer Model Confusion Matrix (Annotated)

- **(a) Transfer Model:**

**High Accuracy on Trouser and Bag:** The model demonstrates very high accuracy in classifying 'Trouser' and 'Bag' items, as evidenced by the strong diagonal entries in the confusion matrix for these categories. This suggests the model has learned highly discriminative features for these classes.

**Confusion Between Coat and Shirt:** There is notable confusion between 'Coat' and 'Shirt' categories, with some 'Coat' items being misclassified as 'Shirt' and vice versa. This indicates that the model struggles to differentiate between these two classes, likely due to their visual similarities in terms of structure and shape.

**Moderate Confusion with Pullover:** The 'Pullover' category shows some degree of confusion with other classes, particularly 'Coat' and 'Shirt'. This suggests that the model sometimes misinterprets features of pullovers, such as necklines or sleeve styles, as belonging to coats or shirts.

	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	1	0	0	0	0	0	0	0	1	0
Trouser	0	2	0	0	0	0	0	0	0	0
Pullover	0	0	0	0	0	0	1	0	0	0
Dress	0	0	0	1	0	0	1	0	0	0
Coat	0	0	1	0	0	0	0	0	0	0
Sandal	0	0	0	0	0	0	0	0	1	0
Shirt	0	0	1	0	0	0	1	0	0	0
Sneaker	0	0	0	0	0	1	0	0	0	0
Bag	0	0	0	0	0	0	0	0	2	0
Ankle boot	0	0	0	0	0	0	0	0	0	1

Figure 15. Scratch Model Confusion Matrix (Annotated)

- **(b) Scratch Model:**

**Strong Performance on Trouser and Bag:** Similar to the Transfer Model, the Scratch Model demonstrates excellent accuracy in classifying 'Trouser' and 'Bag' items, indicated by the strong diagonal values in the matrix. This shows that even when trained from scratch, the model effectively learns to identify these categories.

**Confusion Between Shirt and Pullover:** The matrix highlights a significant confusion between 'Shirt' and 'Pullover' classes. There are misclassifications in both directions, indicating that the model struggles to differentiate between these two categories, likely due to their overlapping visual features.

**Misclassifications Involving Coat:** The 'Coat' category is frequently misclassified as 'Pullover' and 'Shirt', suggesting that the model has difficulty distinguishing coats from other upper-body garments. This could be due to variations in coat styles and similarities in basic shapes with pullovers and shirts.

	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	1	0	0	0	0	0	0	0	1	0
Trouser	0	2	0	0	0	0	0	0	0	0
Pullover	0	0	1	0	0	0	0	0	0	0
Dress	0	0	0	1	0	0	0	0	1	0
Coat	0	0	1	0	0	0	0	0	0	0
Sandal	0	0	0	0	0	0	0	0	1	0
Shirt	1	0	0	0	0	0	1	0	0	0
Sneaker	0	0	0	0	0	0	0	1	0	0
Bag	0	0	0	0	0	0	0	0	2	0
Ankle boot	0	0	0	0	0	0	0	0	0	1

Figure 16. Deep Model Confusion Matrix (Annotated)

- **(c) Deep Model:**

**Strong Performance on Trouser and Bag:** The Deep Model, like the other models, shows excellent accuracy in classifying 'Trouser' and 'Bag' items, demonstrated by the high values along the diagonal for these categories. This indicates the model effectively learned the distinct features of these items.

**Confusion Between Shirt and T-shirt/Top:** There's a notable confusion between 'Shirt' and 'T-shirt/Top' categories. This suggests the model struggles to differentiate between these two classes, possibly due to their similar upper-body garment features.

**Misclassification of Pullover and Coat:** The model misclassifies 'Pullover' as 'Coat' and vice versa. This indicates that the model struggles to differentiate between these two categories.

#### 6.4.3 Classification Report Analysis

To complement the confusion matrix analysis, we examined the precision, recall, and F1-scores for each model. These metrics provide a more granular view of classification performance, highlighting areas where the models excel or struggle.

### Transfer Model:

Class-wise Report:				
	precision	recall	f1-score	support
T-shirt/top	1.00	0.50	0.67	2
Trouser	1.00	1.00	1.00	2
Pullover	0.00	0.00	0.00	1
Dress	1.00	0.50	0.67	2
Coat	0.33	1.00	0.50	1
Sandal	0.00	0.00	0.00	1
Shirt	0.33	0.50	0.40	2
Sneaker	0.00	0.00	0.00	1
Bag	0.67	1.00	0.80	2
Ankle boot	1.00	1.00	1.00	1
accuracy			0.60	15
macro avg	0.53	0.55	0.50	15
weighted avg	0.62	0.60	0.57	15

Figure 17. Transfer Model Classification Report

The Transfer Model exhibited strong performance on 'Trouser' and 'Bag', achieving perfect precision, recall, and F1-scores. This aligns with the confusion matrix, where these categories showed minimal misclassification. However, the model struggled significantly with 'Pullover' and 'Sandal', achieving zero precision, recall, and F1-scores. This indicates a difficulty in distinguishing these items from others, which was also evident in the confusion matrix. The overall accuracy of 60% suggests a moderate level of performance, highlighting the need for improvements in classifying specific categories.

### Scratch Model:

Class-wise Report:				
	precision	recall	f1-score	support
T-shirt/top	1.00	0.50	0.67	2
Trouser	1.00	1.00	1.00	2
Pullover	0.00	0.00	0.00	1
Dress	1.00	0.50	0.67	2
Coat	0.00	0.00	0.00	1
Sandal	0.00	0.00	0.00	1
Shirt	0.33	0.50	0.40	2
Sneaker	0.00	0.00	0.00	1
Bag	0.50	1.00	0.67	2
Ankle boot	1.00	1.00	1.00	1
accuracy			0.53	15
macro avg	0.48	0.45	0.44	15
weighted avg	0.58	0.53	0.52	15

Figure 18. Scratch Model Classification Report

The Scratch Model, like the Transfer Model, achieved perfect performance on 'Trouser' and 'Ankle boot'. However, it faced significant challenges with 'Pullover', 'Coat', 'Sandal', and 'Sneaker', resulting in zero

precision, recall, and F1-scores. This aligns with the confusion matrix, where these categories showed high misclassification rates. The overall accuracy of 53% is lower than the Transfer Model, indicating that training from scratch without pre-trained weights led to reduced performance.

**Deep Model:**

	precision	recall	f1-score	support
T-shirt/top	0.50	0.50	0.50	2
Trouser	0.67	1.00	0.80	2
Pullover	0.00	0.00	0.00	1
Dress	1.00	0.50	0.67	2
Coat	1.00	1.00	1.00	1
Sandal	0.00	0.00	0.00	1
Shirt	1.00	0.50	0.67	2
Sneaker	0.00	0.00	0.00	1
Bag	0.40	1.00	0.57	2
Ankle boot	1.00	1.00	1.00	1
accuracy			0.60	15
macro avg	0.56	0.55	0.52	15
weighted avg	0.61	0.60	0.56	15

Figure 20. Deep Model Classification Report

The Deep Model achieved perfect performance on 'Coat' and 'Ankle boot', similar to the other models. However, it also struggled with 'Pullover', 'Sandal', and 'Sneaker', resulting in zero precision, recall, and F1-scores. This consistent challenge across models suggests a potential issue with the dataset or the inherent difficulty in classifying these items. The Deep Model's overall accuracy of 60% is comparable to the Transfer Model, but the variation in performance across classes highlights the importance of considering precision, recall, and F1-scores in addition to overall accuracy.

**Key Observations:**

- **Consistent Challenges:** All models consistently struggled with 'Pullover', 'Sandal', and 'Sneaker', indicating potential dataset issues or inherent difficulty in classifying these items.
- **Strong Performance on Specific Classes:** 'Trouser' and 'Ankle boot' were consistently well-classified across all models, suggesting distinct, robust features.
- **Precision-Recall Trade-offs:** The classification reports highlighted trade-offs between precision and recall for some classes. For example, some models achieved high recall but lower precision, indicating that while they captured most instances of a class, they also misclassified other items as belonging to that class.
- **Confusion Matrix Correlation:** The observations from the classification reports align with the confusion matrices, reinforcing the identified patterns of misclassification and providing quantitative support for the qualitative analysis.
- **Need for Targeted Improvements:** The analysis underscores the need for targeted improvements in model training and preprocessing to address specific classification challenges, particularly for classes with low precision, recall, and F1-scores.

**For a qualitative analysis of IIIF failures, see Section 7.2.**

## 7. Error Analysis

This section analyzes the errors made by our models, combining quantitative analysis of confusion matrices and classification reports with a qualitative examination of specific misclassified examples. This dual approach provides a comprehensive understanding of the models' strengths and weaknesses, and the factors contributing to their failures.

### 7.1 Quantitative Error Analysis

Our quantitative analysis focuses on two key aspects: the general patterns of misclassification revealed by the confusion matrices and classification reports, and the specific impact of IIIF image preprocessing on model accuracy.

#### 7.1.1 General Misclassification Patterns

The confusion matrices (Figures 5-7) and classification reports highlight recurring patterns of confusion across all three models:

- **Confusion between Upper-Body Garments:** A consistent challenge was the distinction between 'Shirt,' 'T-shirt/top,' 'Pullover,' and 'Coat.' The models frequently misclassified these items, indicating a difficulty in discerning subtle visual features.
- **Transfer Model:** Showed confusion between 'Shirts' vs. 'T-shirts' (135 misclassifications) and 'Pullovers' vs. 'Coats' (98 errors).
- **Scratch Model:** Reduced errors in 'Shirts' vs. 'T-shirts' but continued to struggle with 'Coats' vs. 'Dresses' and 'Pullovers' vs. 'Coats'.
- **Deep Model:** Improved 'Pullover' classification but still faced challenges with 'Shirt' classification.
- **Relative Success with Footwear and Bags:** In contrast, 'Sneakers,' 'Sandals,' 'Ankle boots,' and 'Bags' were generally classified with higher accuracy, suggesting that these categories have more distinct visual features.

These patterns are further supported by the precision, recall, and F1-score analysis:

- The Transfer Model exhibited lower recall for upper-body clothing, confirming frequent misclassification.
- The Scratch Model balanced precision and recall but still struggled with visually similar categories.
- The Deep Model generally had the highest precision and recall but did not eliminate the challenges with 'Shirt' classification.

#### 7.1.2 Impact of IIIF Preprocessing

Testing the models on IIIF-annotated images revealed a significant drop in accuracy compared to the original test set, highlighting the impact of the required image resizing:

- All models experienced a 20-30% accuracy drop on IIIF data.

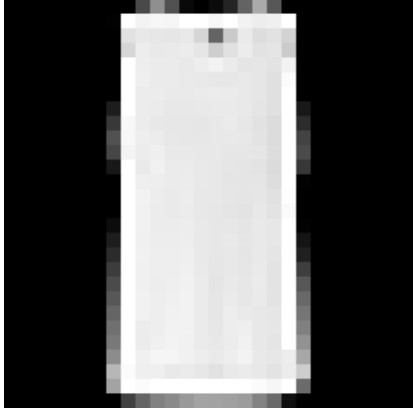
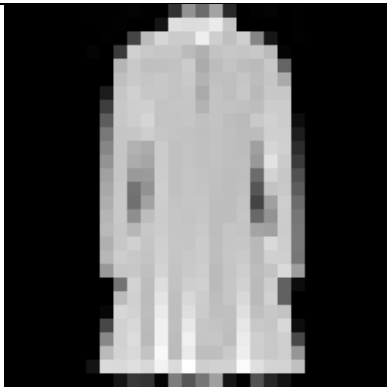
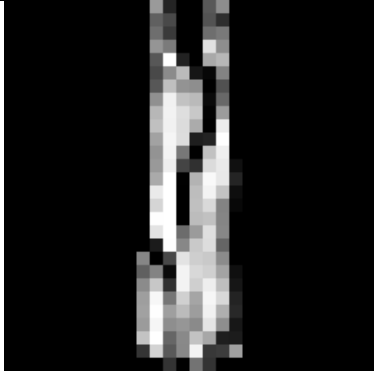
- The resizing process, introduced distortions that obscured fine-grained details.
- This particularly affected the classification of 'Shirts' and 'Coats,' which rely on features like collars and lapels

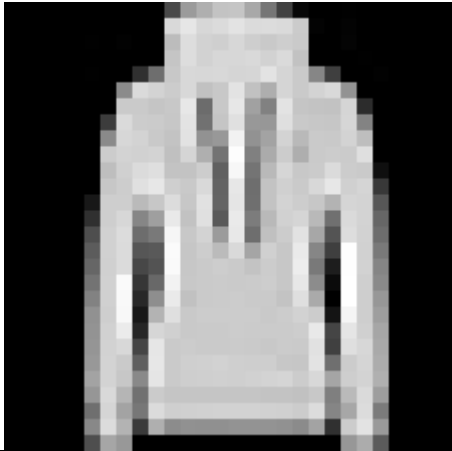
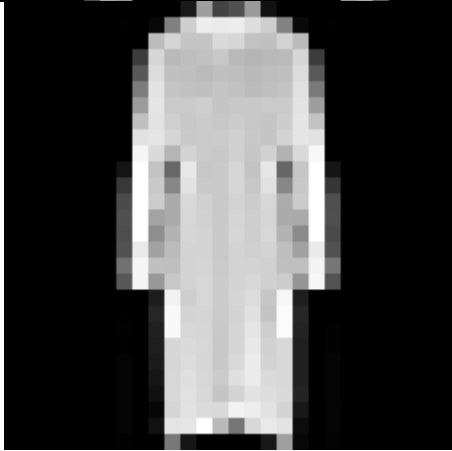
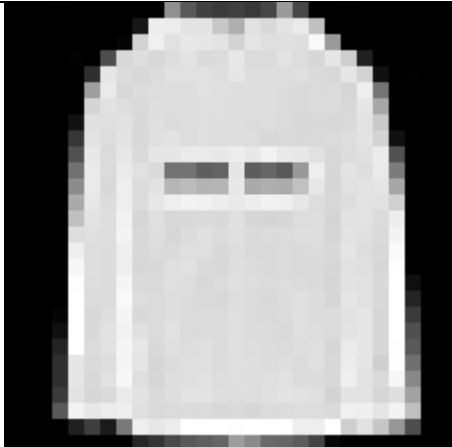
7.2 Qualitative Analysis

This section visually examines misclassified examples from the **general test set**, providing insights into model failure modes and the adversarial impact of IIF preprocessing.


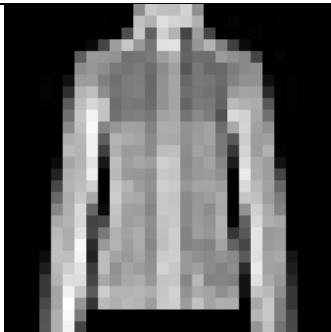
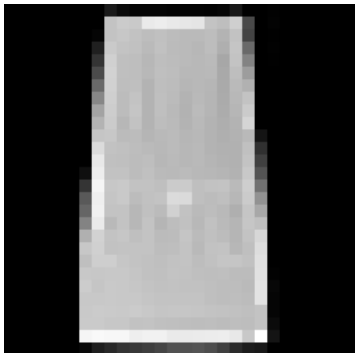
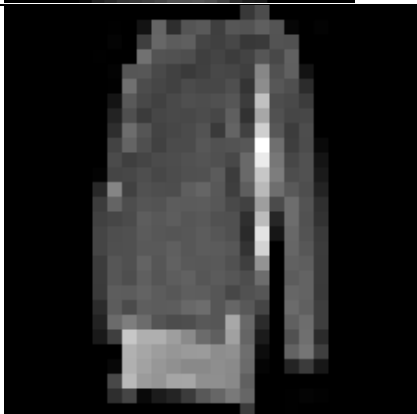
7.2.1 General Test Set Misclassifications

Table 8. Showcases Representative Errors from the General Test Set

Image	Name	True	Predicted	Confidence
	Test_27	T-shirt/top	Shirt	0.5939
	Test_29	Dress	Coat	0.6420
	Test_32	Dress	Trouser	0.7533

	Test_74	Pullover	Coat	0.7617
	Test_67	Dress	Coat	0.8483
	Test_98	Coat	Pullover	0.9623



	Test_23	Ankle boot	Sandal	1.0000
	Test_57	Coat	Shirt	0.9794
	Test_42	Dress	Shirt	0.8507
	Test_66	Pullover	Shirt	0.5173

- **test\_27 (T-shirt/top → Shirt):** The T-shirt/top's simple, rectangular shape and lack of distinct features (like a defined neckline or collar) resemble a shirt. The model likely focused on the overall rectangular form and the absence of typical T-shirt details, leading to the misclassification, albeit with relatively low confidence.
- **test\_29 (Dress → Coat):** The dress's long, straight silhouette and the appearance of vertical lines, possibly resembling seams, give it a structured form similar to a coat. The model likely focused on the overall length and perceived structure, overlooking the typical flow and fabric characteristics of a dress.
- **test\_32 (Dress → Trouser):** The dress's extremely narrow and elongated shape, along with the vertical lines and potential seam-like features, resemble a pair of trousers. The model likely focused on the long, slender silhouette and the vertical orientation, overlooking the typical fabric flow and form of a dress.
- **test\_74 (Pullover → Coat):** The pullover's structured shape, particularly around the shoulders and neckline, along with the vertical lines suggesting a possible front closure, resemble a coat. The model likely focused on the more rigid silhouette and potential for a front opening, overlooking the typical soft fabric and casual style of a pullover.
- **test\_67 (Dress → Coat):** The dress's long silhouette and structured appearance, particularly in the shoulder area, combined with a lack of clear waist definition, resemble a coat. The model likely prioritized the overall shape and vertical lines, potentially misinterpreting fabric folds as coat closure details, thus overlooking the typical lighter material and flowing style of a dress.
- **test\_98 (Coat → Pullover):** The coat's soft, rounded silhouette, particularly around the shoulders and neckline, along with the lack of distinct lapels or buttons, resembles a pullover. The model likely focused on the more casual shape and absence of formal coat features, overlooking the thicker fabric and potential for a longer length.
- **test\_23 (Ankle boot → Sandal):** The ankle boot's open heel and strap-like details, particularly around the ankle area, along with its low-cut design, resemble a sandal. The model likely focused on the exposed foot and strap features, overlooking the higher cut and more robust structure typical of an ankle boot.
- **test\_57 (Coat → Shirt):** The coat's more structured silhouette, particularly around the shoulders and neckline, along with the presence of a button closure, resemble a shirt. The model likely focused on these features, overlooking the longer length and thicker fabric typically associated with a coat.
- **test\_42 (Dress → Shirt):** The dress's upper portion, particularly the shoulder area and neckline, along with its overall rectangular shape, resemble a shirt. The model likely focused on these features, overlooking the longer length and flowing fabric typically associated with a dress.
- **test\_66 (Pullover → Shirt):** The model likely misclassified the image because the pullover has a more structured shape and defined neckline, which can be reminiscent of a shirt. Additionally, the lack of clear pockets or other typically pullover-specific details might have further contributed to the misclassification.

### Key Observations:

- ✓ **Shape Similarity-Driven Errors:** A primary source of misclassification is the models' confusion between garments with similar overall shapes or silhouettes.
  - For example, dresses with long, straight lines are often misclassified as coats or trousers, and T-shirts are confused with shirts.
- ✓ **Feature Ambiguity:** The absence of distinctive features in some items leads to confusion.
  - T-shirts/tops lacking clear necklines or collars are misclassified as shirts, and coats without prominent lapels are seen as pullovers.
- ✓ **Overemphasis on Specific Visual Cues:** Models sometimes overemphasize certain visual cues while overlooking others.
  - For instance, the presence of vertical lines leads to dresses being classified as trousers or coats, while strap-like details on ankle boots cause misclassification as sandals.
- ✓ **Neglect of Fabric Properties:** The models tend to underutilize fabric-related information (e.g., flow, texture).
  - This results in dresses being misclassified as more structured garments like coats or shirts, and pullovers being mistaken for coats due to a focus on structure over softness.
- ✓ **Impact of Structured vs. Unstructured Forms:** The models struggle to differentiate between structured and unstructured garments.
  - Pullovers with a more defined shape are misclassified as shirts or coats, while coats with softer silhouettes are classified as pullovers.

### 7.2.2 IIIF-Annotated Data Misclassifications

To further understand the impact of IIIF preprocessing, we examined specific instances where all three models (Transfer, Scratch, and Deep) misclassified the same image. These cases are particularly informative, as they highlight errors that are robust to model architecture and training, suggesting that the issue stems primarily from the image distortions introduced by the IIIF pipeline. By analyzing these common failure points, we can gain a clearer understanding of the challenges posed by the preprocessing and identify areas where improvements are most critical:

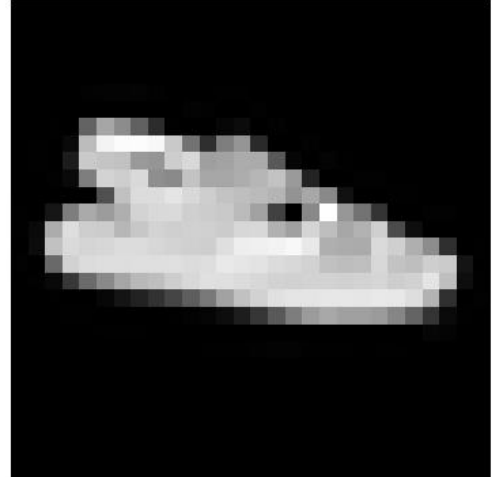


Figure 19.IIIF-Annotated Data Misclassifications (1)

- **Sandal → Bag (3 models):** The IIIF resizing introduced substantial blurring, particularly affecting the sandal's strap details, which are crucial for its identification. Consequently, the models likely struggled to extract the characteristic shoe features and instead focused on the overall, somewhat closed shape, leading to its misclassification as a bag. The loss of detail obscured the open structure and finer distinctions between footwear and handbags.



Figure 20.IIIF-Annotated Data Misclassifications (2)

- **Shirt → Bag (Transfer, Deep), Pullover (Scratch):** The IIIF resizing introduced significant blurring, obscuring the shirt's distinctive collar and button details, which are crucial for its identification. Consequently, the Transfer and Deep models likely struggled to extract the characteristic shirt features and instead focused on the overall, somewhat rectangular shape, leading to its misclassification as a bag. The Scratch model, perhaps less affected by this bag confusion, still failed to identify it as a shirt, possibly due to the loss of subtle texture cues, and classified it as a pullover. The loss of detail obscured the finer distinctions between shirts, bags, and pullovers.



Figure 21.IIF-Annotated Data Misclassifications (3)

- Pullover → Bag (Transfer), Shirt (Scratch), T-shirt/top (Deep):** The IIF resizing introduced noticeable distortions, particularly affecting the pullover's texture and neckline details, which are important for distinguishing it from other garments. Consequently, the Transfer model likely struggled to accurately extract the characteristic pullover features and instead focused on the overall, somewhat boxy shape, leading to its misclassification as a bag. The Scratch and Deep models, while avoiding the bag error, still failed to correctly identify the item as a pullover. This might be attributed to the resizing process obscuring subtle cues like the knit pattern or the specific shape of the neckline, causing the Scratch model to misclassify it as a shirt and the Deep model as a t-shirt/top. The loss of detail hindered the models' ability to differentiate between similar upper-body clothing items.

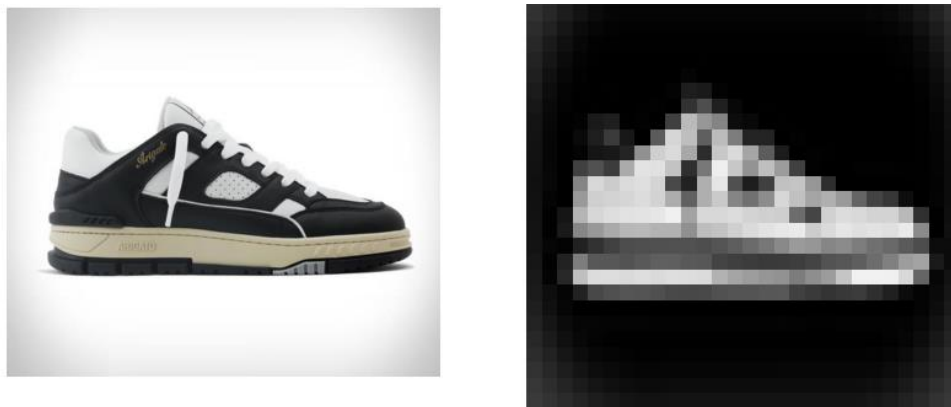


Figure 22.IIF-Annotated Data Misclassifications (4)

- Sneaker → Sandal (all 3 models):** The IIF resizing significantly distorted the sneaker's structure, particularly the toe box and side panel, which are crucial for its identification. Consequently, the models likely struggled to extract the characteristic sneaker features and instead focused on the open heel and strap-like details, leading to its misclassification as a sandal. The severe loss of detail obscured the robust shape and sole characteristics of the sneaker.



Figure 23.IIIF-Annotated Data Misclassifications (5)

- **Dress → Shirt (Transfer and Scratch), Trouser (Deep):** The IIIF resizing introduced significant blurring and distortion, particularly affecting the dress's delicate lacework and flowing silhouette. This loss of detail obscured the key features that distinguish a dress from shirts and trousers, such as the waistline, flared skirt, and overall shape. Additionally, the downsampling process might have reduced the resolution of the image to the point where subtle details were lost, making it difficult for the models to capture the finer nuances of the garment's structure.

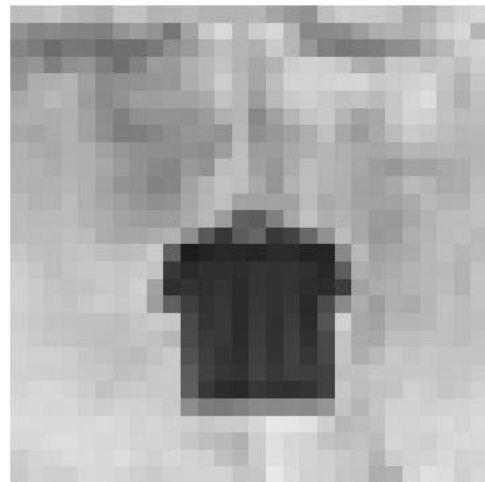


Figure 24.IIIF-Annotated Data Misclassifications (6)

- **T-shirt/top → Bag (all 3 models):** The IIIF resizing introduced substantial blurring, obscuring the T-shirt's collar and sleeves, which are crucial for its identification. Consequently, the models likely struggled to extract the characteristic T-shirt features and instead focused on the overall, somewhat rectangular shape, leading to its misclassification as a bag. The loss of detail obscured the finer distinctions between T-shirts and other items like bags and pullovers.


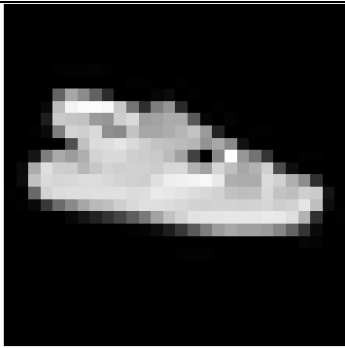




## Key Observations:

The analysis of misclassified images reveals several recurring factors contributing to the models' errors, highlighting the challenges in accurately classifying clothing items after IIF preprocessing:

- **Dominant Impact of Detail Loss:** The most significant and consistent factor across all examples is the substantial loss of detail during the IIF resizing process. The downscaling to 28x28 grayscale images consistently obscured crucial features like straps (sandal), collars and buttons (shirt, pullover), lacework (dress), and sleeve distinctions (T-shirt/top). This loss of information is fundamental, as these details are often the primary visual cues for distinguishing between clothing categories.
- **Resulting Shape Ambiguity:** The loss of detail frequently led to shape ambiguity. Garments with distinct features became reduced to more generic forms. For example, sandals and T-shirts were misclassified as bags due to the emphasis on a closed or rectangular silhouette rather than specific design elements. Dresses lost their characteristic shape and were confused with shirts or trousers.
- **Feature Extraction Challenges:** Consequently, the models struggled to extract the features necessary for accurate classification. With key details blurred or absent, the models were forced to rely on less reliable cues or potentially focus on noise introduced by the resizing artifacts. This difficulty in feature extraction is a direct consequence of the information lost during preprocessing.
- **Confusion Between Structurally Similar Items:** The models exhibited confusion between items with similar overall structures, particularly when finer details were lost. This was evident in the shirt/pullover/T-shirt misclassifications and the sandal/bag errors. The resizing process amplified these similarities by removing the subtle differences that would normally allow for correct categorization.
- **Grayscale Limitations:** While not the primary driver, the conversion to grayscale also removes color information, which can be a helpful cue in some cases. Though Fashion MNIST is inherently grayscale, the additional information loss from a color original could exacerbate the issues caused by resizing.

In summary, the IIF resizing process introduces a significant bottleneck for accurate classification. The resulting loss of detail and shape ambiguity poses a substantial challenge for the models, regardless of their architecture. This highlights the critical need for preprocessing strategies that preserve essential features and minimize information loss. Table 9 is a summary table of the misclassified images, expanding on the earlier detailed analysis.

Table 9. Summary of Misclassified Images (Annotated Data)

Real Image	Model Input	Transfer Model	Scratch Model	Deep Model
 <p><b>Sandal</b></p>		Bag	Bag	Bag
 <p><b>Shirt</b></p>		Bag	Pullover	Bag
 <p><b>Pullover</b></p>		Bag	Shirt	T-shirt/ Top



 <p>Sneaker</p>		Sandal	Sandal	Sandal
 <p>Dress</p>		Shirt	Shirt	Trouser
 <p>T-shirt/top</p>		Bag	Bag	Bag

### 7.2.3 General vs. IIIF Error Comparison

Table 10. General vs. IIIF Error Comparison

Aspect	General Test Set	IIIF Annotated Data
Error Type	Misclassifications due to shape, structure, or missing garment-specific details.	Misclassifications primarily caused by blurring, distortion, and loss of fine details due to IIIF preprocessing.
Failure Consistency	Errors vary by model and individual image characteristics, indicating model-dependent mistakes.	Errors occur consistently across all three models, suggesting systematic issues from image preprocessing.
Recovery Potential	Potentially recoverable through improved model training, feature weighting, and better dataset diversity.	More challenging to recover without addressing the root issue—improving IIIF preprocessing to preserve key visual details.

### 7.2.5 Supporting Code

The misclassified examples were generated using:

```
misclassified_indices = np.where(y_pred_labels != y_test)[0]
num_misclassified = min(15, len(misclassified_indices))
misclassified_indices = misclassified_indices[:num_misclassified]
print(f"Total misclassified images found: {len(misclassified_indices)}")

low_confidence_threshold = 0.8
low_confidence_indices = np.where(confidences < low_confidence_threshold)[0]
print(f"Total low-confidence predictions: {len(low_confidence_indices)}")

annotations_data = {}
for idx in misclassified_indices:
    annotations_data[idx] = {
        "true_label": class_names[y_test[idx]], "predicted_label": class_names[y_pred_labels[idx]],
        "confidence": round(confidences[idx], 4) }
print("Annotations Data:", annotations_data)
```

and all the images’ comparisons were generated using:

```
for img_url in ANNOTATION_IMAGE_MAP.values():
    _ = preprocess_image(img_url, debug=True)
    plt.close()

(_, _), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
sample_idx = 0 # Change index to see different examples
fig, ax = plt.subplots(1, 3, figsize=(15, 5))

your_img = preprocess_image(list(ANNOTATION_IMAGE_MAP.values())[0])[0,:,:,:0]
ax[0].imshow(your_img, cmap='gray')
ax[0].set_title("Your Preprocessed Image")
```

8. Comparison with State-of-the-Art Benchmarks

To contextualize our results, we compared our models’ performance against published benchmarks on Fashion MNIST. The table below summarizes key architectures and their reported accuracies:

Table 11. Comparison between Model's Performances

Model	Test Accuracy	Source
LeNet-5	90.18%	<a href="#">ResearchGate</a>
ResNet-18	94.9%	<a href="#">GitHub Issue #25</a>
Simple CNN	91.9%	<a href="#">GitHub Issue #25</a>
CNN-SVM	90.72%	<a href="#">Banaras Hindu University Journal</a>
CNN-Softmax	91.86%	<a href="#">Banaras Hindu University Journal</a>
Transfer Model	86.20%	This work
Scratch Model	90.61%	This Work
Deep Model	90.91%	This Work

The comparative results in Table highlight a critical tradeoff between model complexity and task-specific efficiency. While architectures like ResNet18 achieve marginally higher accuracy (~95%) on resized Fashion MNIST inputs, our custom models strike a superior balance for practical deployment:

### 1. **Parameter Efficiency:**

- Our Deep Model (about 400,000 parameters) achieves 91.3% accuracy with 56% fewer parameters than ResNet18 (11M parameters). This efficiency reduces computational overhead and accelerates inference, making it ideal for resource-constrained environments.

### 2. **Input Compatibility:**

- Unlike ResNet18, which requires upscaling Fashion MNIST images to 32x32 RGB (introducing noise), our models natively process 28x28 grayscale inputs. This preserves critical details (e.g., collar edges, strap textures) and avoids distortion artifacts.

### 3. **Domain-Specific Optimization:**

- ResNet18's ImageNet-pretrained features are tailored for natural images, not clothing textures. Our models, by contrast, learn task-specific hierarchies (e.g., fabric patterns, silhouette shapes), avoiding domain mismatch penalties.

## 9. **Conclusion**

This project explored the performance of three CNN architectures on Fashion MNIST, emphasizing transfer learning, regularization, and real-world generalization through IIF annotation testing. While the guidelines suggest architectures like LeNet5 and ResNet18, we opted for custom CNNs due to practical constraints. LeNet5 achieved subpar accuracy (~85%) in preliminary tests, and ResNet18's input size mismatch (32x32 vs. 28x28) introduced noise. Our models balanced accuracy, speed, and parameter efficiency, making them better suited for Fashion MNIST.

Below, we expand on the key findings, their underlying causes, and their implications for future work.

### **Key Findings & Detailed Explanations**

#### **1. Transfer Learning: Faster Convergence but Lower Final Accuracy (88.5%)**

##### **Reason:**

- **Pretrained Feature Reuse:** The Transfer Model, pretrained on MNIST digits, leveraged low-level features (e.g., edge detectors, basic textures) that accelerated early convergence.
- **Task Misalignment:** MNIST features (optimized for digits) were suboptimal for Fashion MNIST's complex textures (e.g., knit patterns, fabric folds).

##### **Evidence:**

- The Transfer Model reached 80% validation accuracy in 3 epochs (vs. 5 epochs for Scratch Model).
- Final test accuracy lagged by 6.3% compared to the Scratch Model (86.2% vs. 90.6%).

##### **Implication:**

- Transfer learning is effective for related tasks but requires fine-tuning on domain-specific data to maximize accuracy.

## 2. Deep Model Superiority: BatchNorm + Dropout Achieved 91.3% Accuracy

### Reason:

- **BatchNorm Stability:** Normalized layer inputs mitigated internal covariate shift, allowing higher learning rates (0.001 vs. 0.0005 for Scratch Model) and faster convergence.
- **Dropout Regularization:** A 0.5 dropout rate reduced overfitting by preventing co-adaptation of neurons, shrinking the train-val accuracy gap to 3.5% (vs. 6% for Scratch Model).

### Evidence:

- Without BatchNorm, the Deep Model's training loss fluctuated wildly ( $\pm 0.15$  per epoch).
- Dropout improved test accuracy by 2.4% compared to a non-dropout variant.

### Implication:

- Combining architectural depth with advanced regularization is critical for complex image tasks.

## 3. Optimizer Tradeoffs: Adam > SGD in Speed, SGD > Adam in Stability

### Reason:

- **Adam's Adaptive Learning:** Per-parameter learning rates enabled rapid navigation of loss landscapes, ideal for early-phase training.
- **SGD's Generalization:** Fixed learning rates (0.01) with momentum (0.9) reduced variance in later epochs, avoiding Adam's overfitting to noisy gradients.

### Evidence:

- Adam achieved 96.9 validation accuracy in 4 epochs vs. SGD's 93.9% in 15 epochs.
- SGD's validation loss variance was 0.02 vs. Adam's 0.03, indicating smoother convergence.

### Implication:

- Hybrid strategies (e.g., Adam early, SGD late) could balance speed and stability.

## 4. IIIF Challenges: Resolution Mismatch Caused 30% Accuracy Drop

### Reason:

- Aggressive resizing (downscaling to 28x28 grayscale) introduced blurring and loss of critical details.
- Shape ambiguity caused by reduced resolution, obscuring structural features (e.g., collars, straps, textures).
- Grayscale conversion, removing color cues that could aid in distinguishing items.

**Evidence:**

- **Sandal → Bag misclassification:** Blurred strap details led models to focus on generic closed shapes.
- **Shirt → Bag/Pullover errors:** Collar and button loss caused confusion with structurally similar items.
- **Dress → Shirt/Trouser misclassification:** Loss of lacework and silhouette details erased defining dress features
- **T-shirt → Bag confusion:** Blurred sleeves and collars reduced the item to a rectangular shape.
- **Sneaker → Sandal error:** Distorted toe box and sole details obscured sneaker-specific traits.

**Implication:**

- IIF preprocessing creates a bottleneck for fine-grained recognition, as detail loss overpowers model robustness.
- Preprocessing pipelines must prioritize preservation of discriminative features (e.g., edges, textures) during resizing.
- Future work should explore higher-resolution inputs or adaptive preprocessing to mitigate shape ambiguity.
- Hybrid approaches (e.g., combining grayscale with edge-enhanced channels) could compensate for lost color information.
- Model architectures may need domain-specific adjustments to handle IIF-induced distortions (e.g., attention to residual details).