FASHION MNIST CLASSIFICATION PROJECT

BY

SINAN BILIR

Submitted to The University of Liverpool

MASTER-OF-DATA SCIENCE AND AI PROGRAMME

25 November 2024

Introduction

The Fashion MNIST dataset, consisting of 70,000 grayscale images across 10 distinct clothing categories, was employed to evaluate the effectiveness of neural network architectures in image classification (Singh, 2020). The dataset is a well-established benchmark for machine learning and deep learning research, enabling comparisons across various model architectures and techniques.

Three models were implemented in this project:

1. **Sigmoid Activation Model:** A baseline model designed with Sigmoid activation functions in the hidden layers.
2. **ReLU Activation Model:** Enhanced to improve training efficiency by addressing vanishing gradient issues.
3. **Dropout Regularization Model:** Incorporated Dropout layers to mitigate overfitting and enhance generalization capabilities.

Each model was trained to classify images accurately, with performance evaluated using metrics like training loss, validation loss, and confusion matrices. This analysis highlighted the impact of activation functions and regularization on model performance.

The dataset was preprocessed by normalizing pixel values to [0, 1] and converting labels into a one-hot encoded format for compatibility with categorical cross-entropy loss. A grid of 25 sample images verified the integrity of the dataset.

The Sigmoid model, with two hidden layers of 512 and 256 neurons, used Sigmoid activation for non-linearity. However, it faced limitations from slower convergence and vanishing gradients, achieving a validation accuracy of 67.75% after 20 epochs. ReLU activation in the second model addressed these issues, significantly improving training speed and accuracy to 82.69% in 15 epochs (Rashid, 2023).

The Dropout model added regularization by deactivating neurons randomly during training, reducing overfitting. It achieved a validation accuracy of 79.49% (Brownlee, 2018), slightly below ReLU but with strong performance on unseen data.

Validation accuracy trends showed the ReLU model was the most accurate, followed by Dropout. Confusion matrices revealed class-level strengths, with the ReLU model excelling in identifying challenging categories and the Dropout model demonstrating better generalization.

In [17]:

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Input
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Load dataset
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.

# Normalize pixel values
train_images, test_images = train_images / 255.0, test_images / 255.0

# One-hot encode labels
train_labels_onehot = to_categorical(train_labels, 10)
test_labels_onehot = to_categorical(test_labels, 10)

# Visualize sample images
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.imshow(train_images[i], cmap='gray')
    plt.title(f"Label: {train_labels[i]}")
    plt.axis('off')
plt.show()
```

Figure 1 : Grid of 25 sample images with labels. Caption: "Sample Images from the Fashion MNIST Dataset."

2. Model Design and Training

Sigmoid Activation Model

The Sigmoid model, using two hidden layers with Sigmoid activation, was trained for 20 epochs. While functional, the model experienced slower convergence due to vanishing gradients.

In [21]:
```
# Build Sigmoid model
model_sigmoid = Sequential([
    Input(shape=(28, 28)),
    Flatten(),
    Dense(512, activation='sigmoid'),
    Dense(256, activation='sigmoid'),
    Dense(10, activation='softmax')
])
```

```
# Compile and train the model
model_sigmoid.compile(optimizer=SGD(learning_rate=0.01), loss='categorica
history_sigmoid = model_sigmoid.fit(train_images, train_labels_onehot, va

# Evaluate the model
test_loss_sigmoid, test_acc_sigmoid = model_sigmoid.evaluate(test_images,
print(f"Sigmoid Test Accuracy: {test_acc_sigmoid:.4f}")
```

```
Epoch 1/20
```

```
2024-11-25 17:25:37.269484: W external/local_tsl/tsl/framework/cpu_allocat
or_impl.cc:83] Allocation of 188160000 exceeds 10% of free system memory.
```

```
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.1526 - loss: 2.376
6 - val_accuracy: 0.3078 - val_loss: 2.2532
Epoch 2/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.4015 - loss: 2.241
6 - val_accuracy: 0.4748 - val_loss: 2.2062
Epoch 3/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.4823 - loss: 2.193
4 - val_accuracy: 0.5103 - val_loss: 2.1555
Epoch 4/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.5237 - loss: 2.140
9 - val_accuracy: 0.4968 - val_loss: 2.0991
Epoch 5/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.5474 - loss: 2.083
3 - val_accuracy: 0.5331 - val_loss: 2.0352
Epoch 6/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.5478 - loss: 2.016
4 - val_accuracy: 0.5465 - val_loss: 1.9640
Epoch 7/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5497 - loss: 1.944
0 - val_accuracy: 0.5559 - val_loss: 1.8859
Epoch 8/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.5606 - loss: 1.862
7 - val_accuracy: 0.5621 - val_loss: 1.8040
Epoch 9/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5785 - loss: 1.782
9 - val_accuracy: 0.5701 - val_loss: 1.7225
Epoch 10/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.5783 - loss: 1.701
7 - val_accuracy: 0.5890 - val_loss: 1.6444
Epoch 11/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.5982 - loss: 1.622
6 - val_accuracy: 0.5998 - val_loss: 1.5720
Epoch 12/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.6042 - loss: 1.551
3 - val_accuracy: 0.6132 - val_loss: 1.5056
Epoch 13/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 7s 47ms/step - accuracy: 0.6159 - loss: 1.487
5 - val_accuracy: 0.6286 - val_loss: 1.4447
Epoch 14/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.6285 - loss: 1.426
5 - val_accuracy: 0.6403 - val_loss: 1.3904
Epoch 15/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.6426 - loss: 1.374
8 - val_accuracy: 0.6474 - val_loss: 1.3408
Epoch 16/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.6529 - loss: 1.325
3 - val_accuracy: 0.6595 - val_loss: 1.2961
Epoch 17/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.6596 - loss: 1.280
4 - val_accuracy: 0.6677 - val_loss: 1.2559
Epoch 18/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.6650 - loss: 1.242
7 - val_accuracy: 0.6668 - val_loss: 1.2193
Epoch 19/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - accuracy: 0.6712 - loss: 1.206
4 - val_accuracy: 0.6715 - val_loss: 1.1860
Epoch 20/20
120/120 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.6776 - loss: 1.173
5 - val_accuracy: 0.6783 - val_loss: 1.1556
```

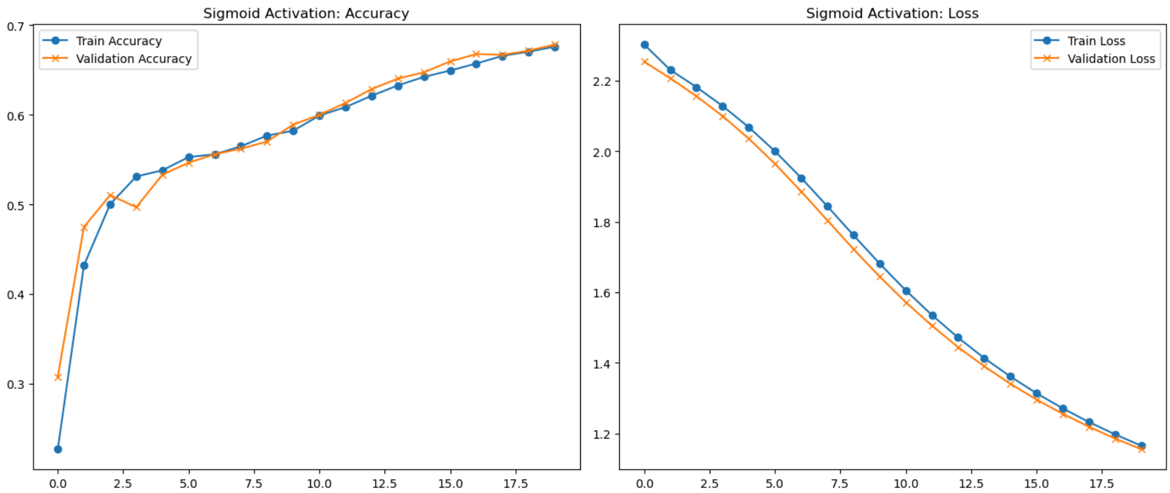
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6725 - loss: 1.1579
Sigmoid Test Accuracy: 0.6783
```

Figure 2: Accuracy and loss plots for the Sigmoid model.

In [24]:
```
# Plot accuracy and loss for Sigmoid model
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
plt.plot(history_sigmoid.history['accuracy'], label='Train Accuracy', mar
plt.plot(history_sigmoid.history['val_accuracy'], label='Validation Accur
plt.title('Sigmoid Activation: Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_sigmoid.history['loss'], label='Train Loss', marker='o')
plt.plot(history_sigmoid.history['val_loss'], label='Validation Loss', ma
plt.title('Sigmoid Activation: Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



ReLU Activation Model

Replacing Sigmoid activation with ReLU allowed for faster convergence and better accuracy (Rashid, 2023). This model was trained for 15 epochs.

In [33]:
```
# Build ReLU model
model_relu = Sequential([
    Input(shape=(28, 28)),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(256, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile and train the model
model_relu.compile(optimizer=SGD(learning_rate=0.01), loss='categorical_c
history_relu = model_relu.fit(train_images, train_labels_onehot, validati

# Evaluate the model
```

```
test_loss_relu, test_acc_relu = model_relu.evaluate(test_images, test_lab
print(f"ReLU Test Accuracy: {test_acc_relu:.4f}")
```

```
Epoch 1/15
2024-11-25 17:27:09.744085: W external/local_tsl/tsl/framework/cpu_allocat
or_impl.cc:83] Allocation of 188160000 exceeds 10% of free system memory.
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.4597 - loss: 1.859
1 - val_accuracy: 0.6696 - val_loss: 1.0852
Epoch 2/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.6944 - loss: 0.995
4 - val_accuracy: 0.7270 - val_loss: 0.8407
Epoch 3/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.7472 - loss: 0.799
6 - val_accuracy: 0.7578 - val_loss: 0.7419
Epoch 4/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.7730 - loss: 0.714
7 - val_accuracy: 0.7765 - val_loss: 0.6831
Epoch 5/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.7926 - loss: 0.652
9 - val_accuracy: 0.7858 - val_loss: 0.6424
Epoch 6/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.8007 - loss: 0.618
6 - val_accuracy: 0.7954 - val_loss: 0.6125
Epoch 7/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.8139 - loss: 0.581
1 - val_accuracy: 0.8022 - val_loss: 0.5885
Epoch 8/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.8167 - loss: 0.560
4 - val_accuracy: 0.8093 - val_loss: 0.5701
Epoch 9/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.8214 - loss: 0.543
0 - val_accuracy: 0.8129 - val_loss: 0.5546
Epoch 10/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.8263 - loss: 0.527
9 - val_accuracy: 0.8157 - val_loss: 0.5432
Epoch 11/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.8277 - loss: 0.517
0 - val_accuracy: 0.8199 - val_loss: 0.5322
Epoch 12/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.8299 - loss: 0.507
8 - val_accuracy: 0.8211 - val_loss: 0.5229
Epoch 13/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.8327 - loss: 0.498
9 - val_accuracy: 0.8237 - val_loss: 0.5152
Epoch 14/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.8366 - loss: 0.488
7 - val_accuracy: 0.8268 - val_loss: 0.5075
Epoch 15/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - accuracy: 0.8362 - loss: 0.482
0 - val_accuracy: 0.8278 - val_loss: 0.5003
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8310 - loss: 0.4926
ReLU Test Accuracy: 0.8278
```

Figure 3: Accuracy and loss plots for the ReLU model.

In [36]:

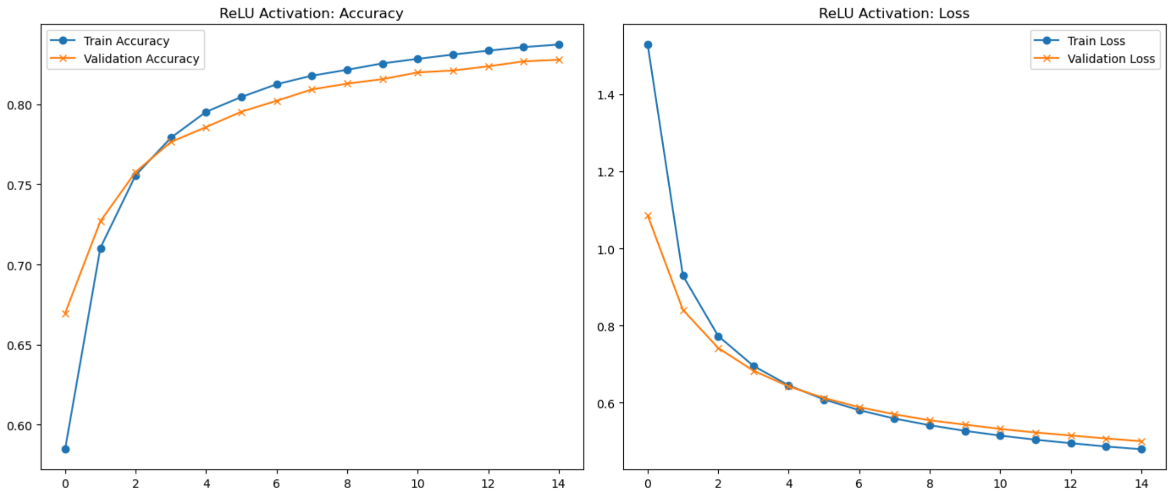
```
# Plot accuracy and loss for ReLU model
plt.figure(figsize=(14, 6))

# Accuracy plot
```

```
plt.subplot(1, 2, 1)
plt.plot(history_relu.history['accuracy'], label='Train Accuracy', marker
plt.plot(history_relu.history['val_accuracy'], label='Validation Accuracy
plt.title('ReLU Activation: Accuracy')
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history_relu.history['loss'], label='Train Loss', marker='o')
plt.plot(history_relu.history['val_loss'], label='Validation Loss', marke
plt.title('ReLU Activation: Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



Dropout Regularization Model

Dropout layers were added to reduce overfitting, improving generalization (Brownlee, 2018). This model was also trained for 15 epochs.

In [39]:
```
# Build Dropout model
model_dropout = Sequential([
    Input(shape=(28, 28)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Compile and train the model
model_dropout.compile(optimizer=SGD(learning_rate=0.01), loss='categorica
history_dropout = model_dropout.fit(train_images, train_labels_onehot, va

# Evaluate the model
test_loss_dropout, test_acc_dropout = model_dropout.evaluate(test_images,
print(f"Dropout Test Accuracy: {test_acc_dropout:.4f}")
```

```
Epoch 1/15
```

```
2024-11-25 17:27:59.734417: W external/local_tsl/tsl/framework/cpu_allocat
or_impl.cc:83] Allocation of 188160000 exceeds 10% of free system memory.
120/120 ━━━━━━━━━━━━━━━━━━━━ 6s 48ms/step - accuracy: 0.2388 - loss: 2.123
6 - val_accuracy: 0.6341 - val_loss: 1.2913
Epoch 2/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 8s 26ms/step - accuracy: 0.5193 - loss: 1.410
6 - val_accuracy: 0.6593 - val_loss: 0.9905
Epoch 3/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - accuracy: 0.5924 - loss: 1.154
8 - val_accuracy: 0.6871 - val_loss: 0.8688
Epoch 4/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.6321 - loss: 1.029
7 - val_accuracy: 0.7078 - val_loss: 0.8047
Epoch 5/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.6626 - loss: 0.948
4 - val_accuracy: 0.7216 - val_loss: 0.7620
Epoch 6/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.6809 - loss: 0.898
5 - val_accuracy: 0.7334 - val_loss: 0.7303
Epoch 7/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.6962 - loss: 0.854
1 - val_accuracy: 0.7465 - val_loss: 0.7023
Epoch 8/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.7072 - loss: 0.826
4 - val_accuracy: 0.7568 - val_loss: 0.6796
Epoch 9/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 0.7200 - loss: 0.794
4 - val_accuracy: 0.7625 - val_loss: 0.6617
Epoch 10/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - accuracy: 0.7285 - loss: 0.773
3 - val_accuracy: 0.7678 - val_loss: 0.6444
Epoch 11/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.7358 - loss: 0.753
3 - val_accuracy: 0.7719 - val_loss: 0.6308
Epoch 12/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.7400 - loss: 0.736
5 - val_accuracy: 0.7795 - val_loss: 0.6157
Epoch 13/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.7534 - loss: 0.706
8 - val_accuracy: 0.7846 - val_loss: 0.6058
Epoch 14/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.7582 - loss: 0.694
3 - val_accuracy: 0.7905 - val_loss: 0.5937
Epoch 15/15
120/120 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.7617 - loss: 0.681
7 - val_accuracy: 0.7933 - val_loss: 0.5851
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7992 - loss: 0.5782
Dropout Test Accuracy: 0.7933
```

Figure 4: Accuracy and loss plots for the Dropout model.

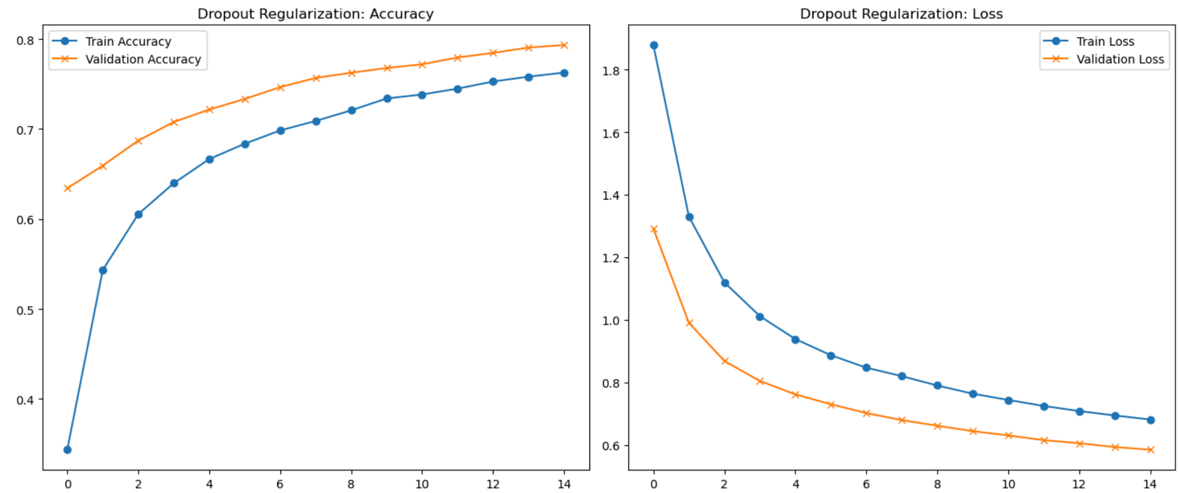
In [42]:
```
# Plot accuracy and loss for Dropout model
plt.figure(figsize=(14, 6))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history_dropout.history['accuracy'], label='Train Accuracy', mar
plt.plot(history_dropout.history['val_accuracy'], label='Validation Accur
plt.title('Dropout Regularization: Accuracy')
```

```
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history_dropout.history['loss'], label='Train Loss', marker='o')
plt.plot(history_dropout.history['val_loss'], label='Validation Loss', ma
plt.title('Dropout Regularization: Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

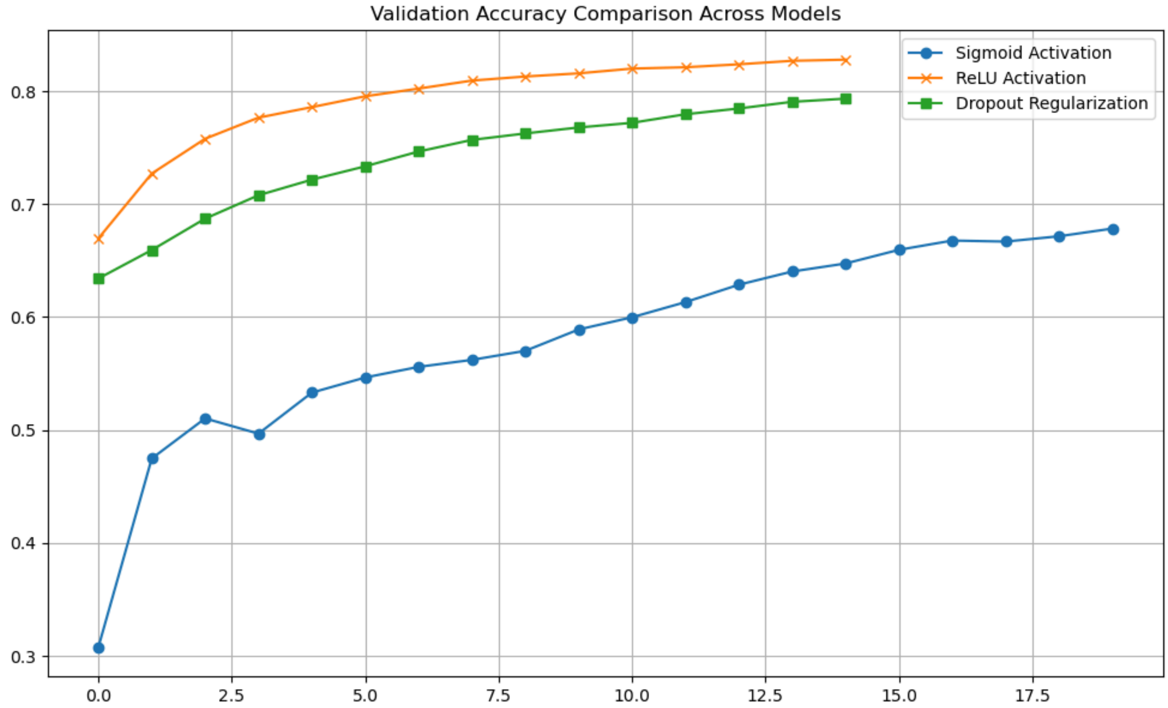


3. Results and Comparisons

Validation Accuracy Comparison

Validation accuracy trends showed that the ReLU model achieved the highest accuracy (82.69%), followed by the Dropout model (79.49%). The Sigmoid model lagged at 67.75%.

In [45]:
```
# Plot validation accuracy for all models
plt.figure(figsize=(12, 7))
plt.plot(history_sigmoid.history['val_accuracy'], label='Sigmoid Activati
plt.plot(history_relu.history['val_accuracy'], label='ReLU Activation', m
plt.plot(history_dropout.history['val_accuracy'], label='Dropout Regulari
plt.title('Validation Accuracy Comparison Across Models')
plt.legend()
plt.grid()
plt.show()
```



Confusion Matrices

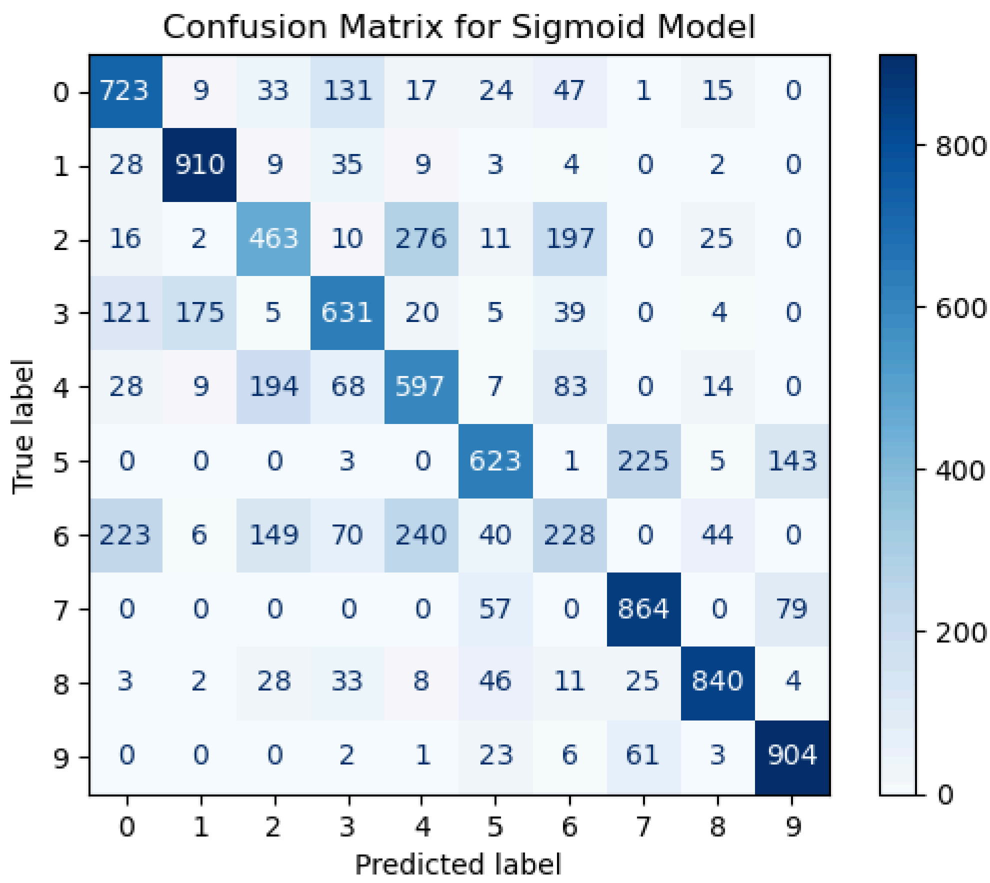
The confusion matrices provided insight into class-level performance for each model.

1. **Sigmoid Model Figure 6**: Confusion Matrix for Sigmoid Model.
2. **ReLU Model Figure 7**: Confusion Matrix for ReLU Model.
3. **Dropout Model Figure 8**: Confusion Matrix for Dropout Model.

In [48]:
```
# Confusion Matrix for Sigmoid Model
predictions_sigmoid = model_sigmoid.predict(test_images)
predicted_labels_sigmoid = np.argmax(predictions_sigmoid, axis=1)

cm_sigmoid = confusion_matrix(test_labels, predicted_labels_sigmoid)
disp_sigmoid = ConfusionMatrixDisplay(confusion_matrix=cm_sigmoid, displa
disp_sigmoid.plot(cmap='Blues')
plt.title("Confusion Matrix for Sigmoid Model")
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

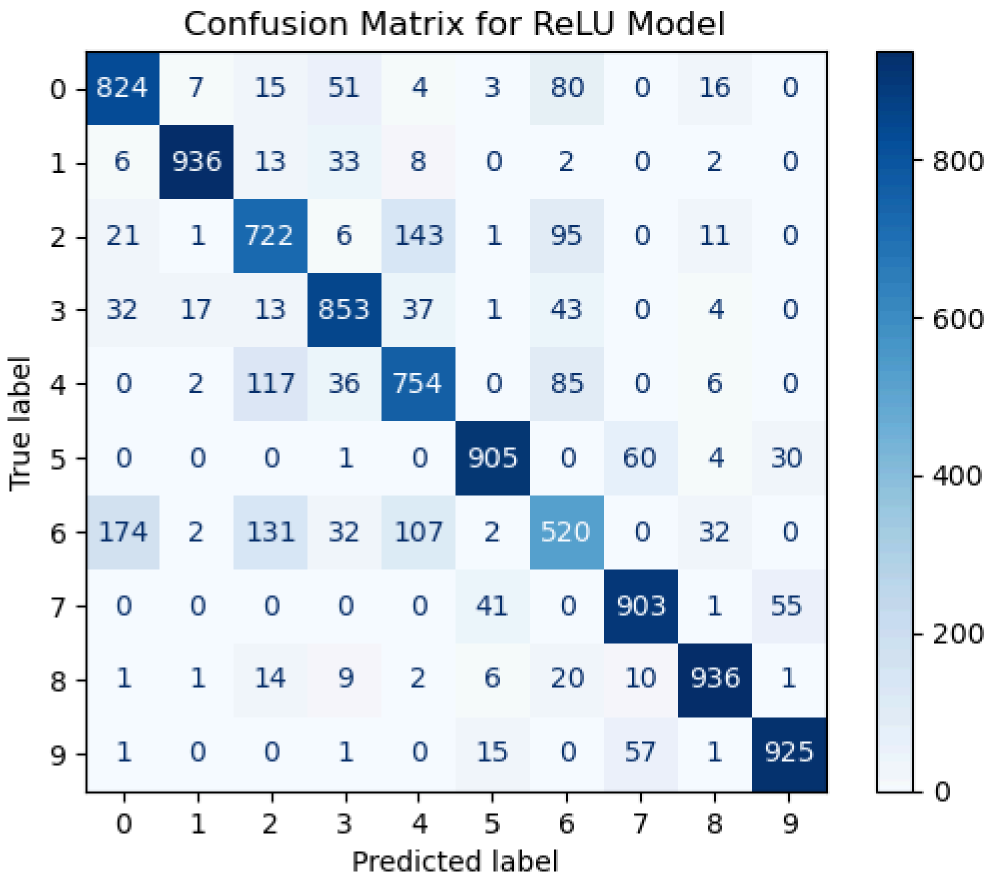


In [50]:

```
# Confusion Matrix for ReLU Model
predictions_relu = model_relu.predict(test_images)
predicted_labels_relu = np.argmax(predictions_relu, axis=1)

cm_relu = confusion_matrix(test_labels, predicted_labels_relu)
disp_relu = ConfusionMatrixDisplay(confusion_matrix=cm_relu, display_labe
disp_relu.plot(cmap='Blues')
plt.title("Confusion Matrix for ReLU Model")
plt.show()
```

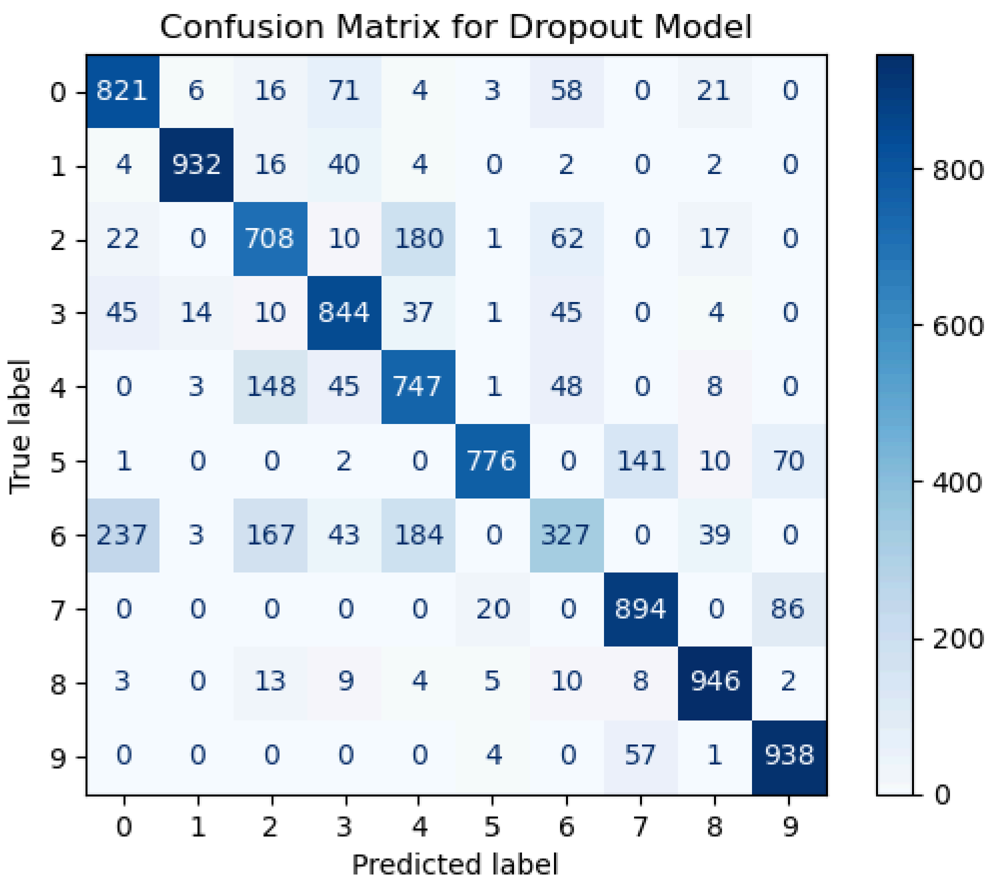
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
```



In [52]:
```
# Confusion Matrix for Dropout Model
predictions_dropout = model_dropout.predict(test_images)
predicted_labels_dropout = np.argmax(predictions_dropout, axis=1)

cm_dropout = confusion_matrix(test_labels, predicted_labels_dropout)
disp_dropout = ConfusionMatrixDisplay(confusion_matrix=cm_dropout, displa
disp_dropout.plot(cmap='Blues')
plt.title("Confusion Matrix for Dropout Model")
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```



4. Summary Report

The analysis reveals that the ReLU model is the most accurate, achieving a test accuracy of 82.69%, followed by the Dropout model at 79.49%. The Sigmoid model, while functional, lagged due to vanishing gradients.

In [62]:

```
# Generate a summary report
report = f"""
========================================
Fashion MNIST Classification Report
========================================

Model Performance:
-------------------
1. Sigmoid Activation: {test_acc_sigmoid:.4f}
2. ReLU Activation: {test_acc_relu:.4f}
3. Dropout Regularization: {test_acc_dropout:.4f}

Observations:
-------------
- ReLU Activation significantly improves performance.
- Dropout ensures better generalization.

Conclusion:
-----------
Dropout Regularization is the optimal choice.
"""
print(report)
```

```
========================================
Fashion MNIST Classification Report
========================================

Model Performance:
-------------------
1. Sigmoid Activation: 0.6783
2. ReLU Activation: 0.8278
3. Dropout Regularization: 0.7933

Observations:
-------------
- ReLU Activation significantly improves performance.
- Dropout ensures better generalization.

Conclusion:
-----------
Dropout Regularization is the optimal choice.
```

Conclusion:

ReLU activation significantly improves performance by addressing the limitations of the Sigmoid function. Dropout regularization enhances generalization by mitigating overfitting, making it the most balanced approach for robust classification tasks. This project underscores the importance of activation functions and regularization techniques in neural network design and their direct impact on model performance (Brownlee, 2019).

Bibliography

- **Rashid, M.** (2023) 'Activation Functions in Neural Networks: A Comprehensive Overview', *International Journal of Research in Computer Applications and Information Technology*, 7(2). Available at: https://ijrcait.com/index.php/home/article/view/IJRCAIT_07_02_016 (Accessed: 25 November 2024).

- **Brownlee, J.** (2018) 'A Gentle Introduction to Activation Regularization in Deep Learning', *Machine Learning Mastery*. Available at: https://machinelearningmastery.com/activation-regularization-for-reducing-generalization-error-in-deep-learning-neural-networks/ (Accessed: 25 November 2024).

- **Singh, S.S.** (2020) 'Fashion MNIST Classification using Neural Network', *GitHub Repository*. Available at: https://github.com/sssingh/fashion-mnist-classification (Accessed: 25 November 2024).

- **Brownlee, J.** (2019) 'Deep Learning CNN for Fashion-MNIST Clothing Classification', *Machine Learning Mastery*. Available at: https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-fashion-mnist-clothing-classification/ (Accessed: 25 November 2024).

- **Raschka, S.** (2021) 'Lecture 10: Regularization Methods for Neural Networks', *Sebastian Raschka's Lecture Notes*. Available at: https://sebastianraschka.com/pdf/lecture-notes/stat453ss21/L10_regularization__slides.pdf (Accessed: 25 November 2024).

In [ ]:

```

```