## Group No

## Group Member Names:

1. Abhinav Gazta | 2023AC05645 | 100%
2. Raghumandala Chaitanya | 2023ac05752 | 100%
3. Kamutam sridharraj | 2023AC05117 | 100%
4. Priyanka Telukuntala | 2023ac05185 | 100%

# 1. Import the required libraries

In [2]:
```
    ##---------Type the code below this line-----------------##
import tensorflow as tf
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import ssl
import pandas as pd
import cv2
from tensorflow import keras
from tensorflow.keras import layers, regularizers
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
```

```
2025-03-09 21:18:42.820666: I tensorflow/core/platform/cpu_feature_guard.c
c:210] This TensorFlow binary is optimized to use available CPU instructio
ns in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebui
ld TensorFlow with the appropriate compiler flags.
```

# 2. Data Acquisition -- Score: 0.5 Mark

For the problem identified by you, students have to find the data source themselves from any data source.

## 2.1 Code for converting the above downloaded data into a form suitable for DL

Deepfashion dataset which is a large-scale clothes dataset consisting of 800,000 images in total. The dataset is divided into 50 categories and each category has 25,000 images. The images are of high resolution and each image is labeled with category, attribute labels, and landmark labels. Also Leveraging the keras dataset which is in built to work on the problem.

## 2.1 Write your observations from the above.

1. Size of the dataset Fashion-MNIST consists of: Training set: 60,000 images Test set: 10,000 images Each image size: 28 × 28 pixels (grayscale)
2. What type of data attributes are there? Images: Grayscale images of clothing items. Each pixel has an intensity value between 0 (black) and 255 (white). Labels: Integer labels (0 to 9) corresponding to different fashion categories
3. What are you classifying? 0: T-shirt/top 1: Trouser 2: Pullover 3: Dress 4: Coat 5: Sandal 6: Shirt 7: Sneaker 8: Bag 9: Ankle boot The task is to classify images into one of these 10 fashion categories.
4. Plot the distribution of the categories of the target / label.

In [3]:
```
ssl._create_default_https_context = ssl._create_unverified_context
# Load the dataset

fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.
```

----------Type the answers below this line----------

# 3. Data Preparation -- Score: 1 Mark

Perform the data prepracessing that is required for the data that you have downloaded. This stage depends on the dataset that is used.

## 3.1 Apply pre-processing techiniques

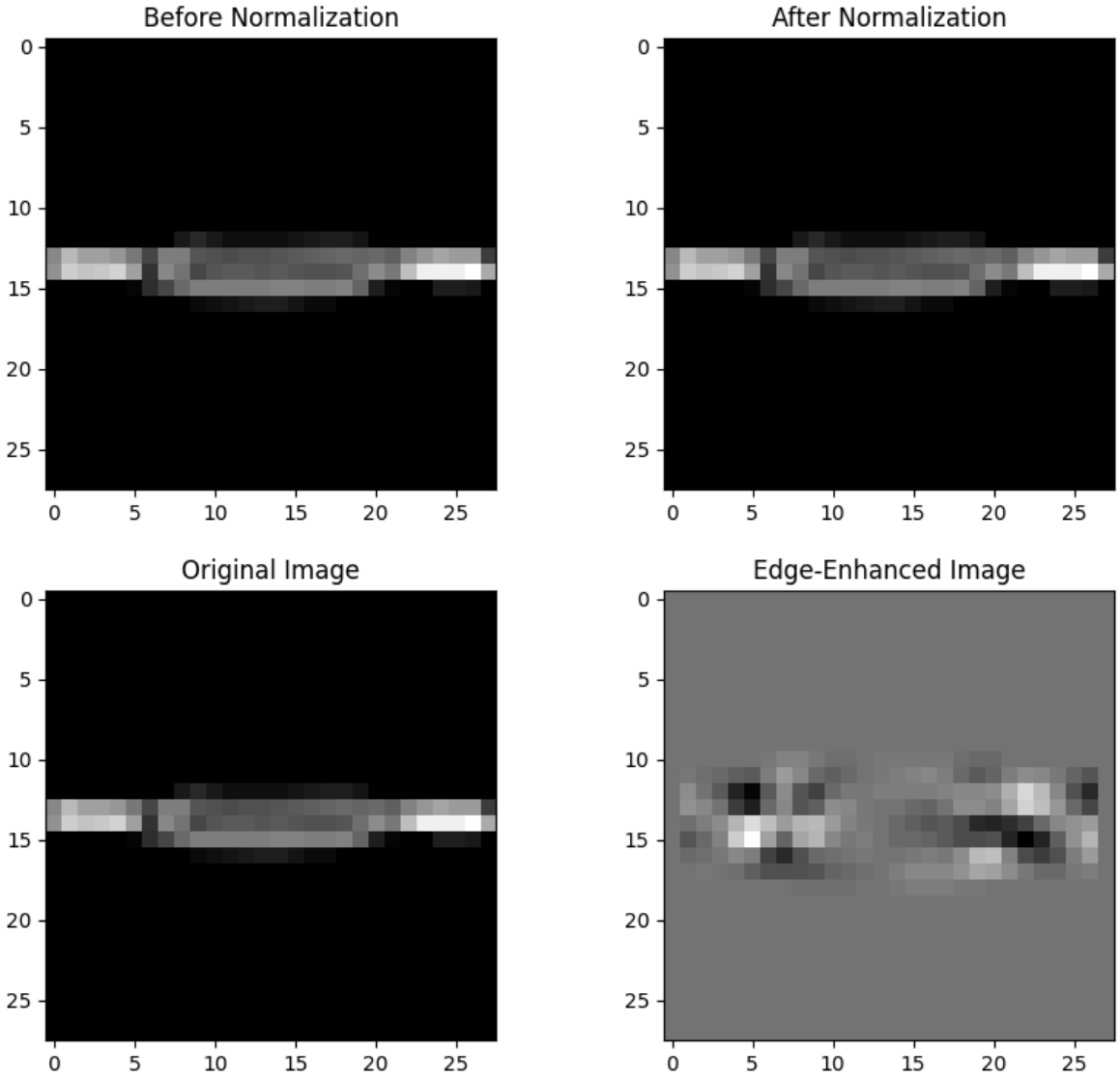
- to remove duplicate data
- to impute or remove missing data
- to remove data inconsistencies
- Encode categorical data
- Normalize the data
- Feature Engineering
- Stop word removal, lemmatiation, stemming, vectorization

IF ANY

In [4]:
```
##---------Type the code below this line-----------------##
## Remove duplicate images from the training set.
# Flatten the images for comparison
flattened_train = train_images.reshape(train_images.shape[0], -1)

# Convert to tuples for hashability
unique_train_images, unique_indices = np.unique(flattened_train, axis=0,

# Filter unique labels
train_images = train_images[unique_indices]
train_labels = train_labels[unique_indices]

print(f"Unique training images: {len(train_images)} (original: {len(flatt

# Check for missing values
```

```
print("Missing values in training images:", np.isnan(train_images).sum())
print("Missing values in test images:", np.isnan(test_images).sum())

# If missing values exist, we replace them with the mean pixel value:
train_images = np.nan_to_num(train_images, nan=np.mean(train_images))
test_images = np.nan_to_num(test_images, nan=np.mean(test_images))

# Normalize images (scale pixel values to 0–1)
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0

# Plot before & after normalization
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4))

# Original image
plt.subplot(1, 2, 1)
plt.imshow(train_images[0] * 255, cmap='gray')  # Multiply by 255 to show
plt.title("Before Normalization")

# Normalized image
plt.subplot(1, 2, 2)
plt.imshow(train_images[0], cmap='gray')
plt.title("After Normalization")

plt.show()


# Apply Sobel filter to enhance edges
edge_image = cv2.Sobel(train_images[0], cv2.CV_64F, 1, 1, ksize=5)

# Plot original vs. edge-detected image
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.imshow(train_images[0], cmap='gray')
plt.title("Original Image")

plt.subplot(1, 2, 2)
plt.imshow(edge_image, cmap='gray')
plt.title("Edge-Enhanced Image")

plt.show()
```

```
Unique training images: 60000 (original: 60000)
Missing values in training images: 0
Missing values in test images: 0
```



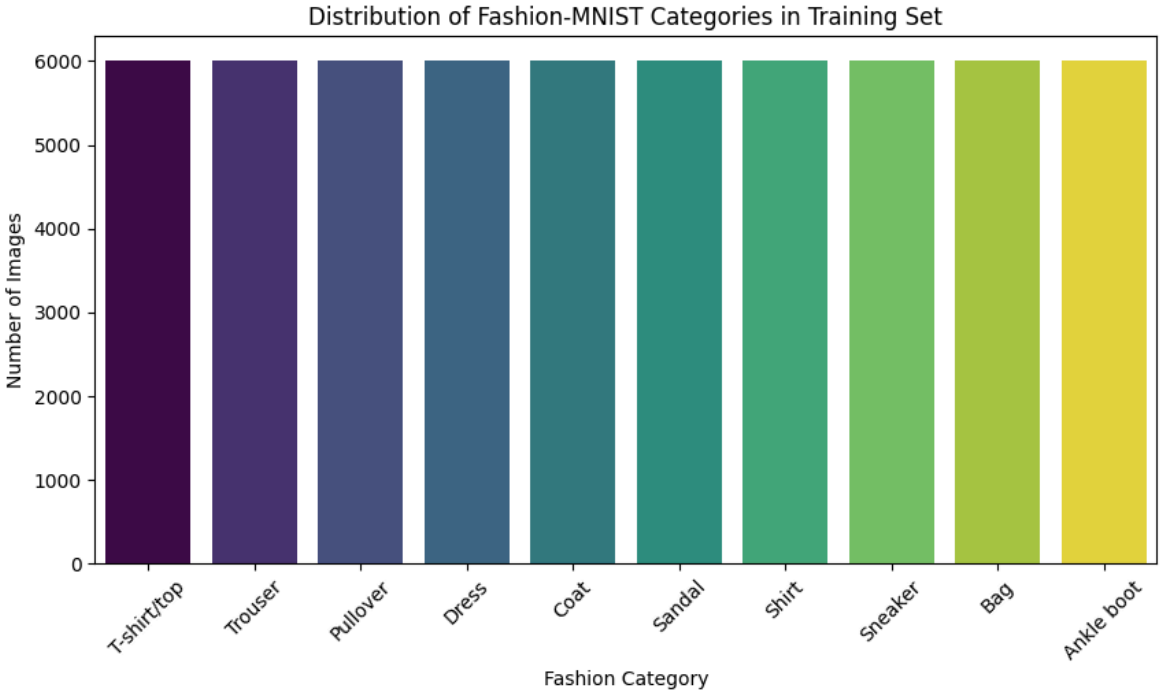

## 3.2 Identify the target variables.

- Separate the data front the target such that the dataset is in the form of (X,y) or (Features, Label)
- Discretize / Encode the target variable or perform one-hot encoding on the target or any other as and if required.

In [5]:
```
##---------Type the code below this line-----------------##

(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
# One-hot encode target labels
y_train_encoded = to_categorical(y_train, num_classes=10)
y_test_encoded = to_categorical(y_test, num_classes=10)

# Print shape after encoding
print("Shape of y_train_encoded:", y_train_encoded.shape)  # (60000, 10)
print("Shape of y_test_encoded:", y_test_encoded.shape)    # (10000, 10)

# Example before & after encoding
print("Before encoding:", y_train[0])  # Example: 9
print("After encoding:", y_train_encoded[0])  # [0,0,0,0,0,0,0,0,0,1]
```

```
# Define class names
class_names = [
    "T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
    "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"
]

# Count occurrences of each class
unique, counts = np.unique(train_labels, return_counts=True)

# Convert to DataFrame for Seaborn

df = pd.DataFrame({"Category": unique, "Count": counts})

# Plot distribution
plt.figure(figsize=(10,5))
sns.barplot(data=df, x="Category", y="Count", hue="Category", palette="vi
plt.xticks(ticks=unique, labels=class_names, rotation=45)
plt.xlabel("Fashion Category")
plt.ylabel("Number of Images")
plt.title("Distribution of Fashion-MNIST Categories in Training Set")
plt.show()
```

```
Shape of y_train_encoded: (60000, 10)
Shape of y_test_encoded: (10000, 10)
Before encoding: 9
After encoding: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
```




## 3.3 Split the data into training set and testing set

In [6]:
```
##---------Type the code below this line-----------------##
# Print shapes
print("Training set shape:", train_images.shape)  # (60000, 28, 28)
print("Test set shape:", test_images.shape)      # (10000, 28, 28)

# Print new shapes
print("New Training set shape:", X_train.shape)  # (48000, 28, 28)    # (
print("Test set shape:", test_images.shape)      # (10000, 28, 28)
```

```
Training set shape: (60000, 28, 28)
Test set shape: (10000, 28, 28)
New Training set shape: (60000, 28, 28)
Test set shape: (10000, 28, 28)
```

In [7]:
```
##---------Type the answer below this line-----------------##
"""
Fashion-MNIST is a dataset of Zalando's article images consisting of a tr
'and a test set of 10,000 examples.
 Each example is a 28x28 grayscale image, associated with a label from 10
 """
# TRAINING SET SIZE = 60000 samples.
# TESTING SET SIZE = 10000 samples.
# IMAGE SIZE = 28x28 pixels.
# IMAGE TYPE = grayscale.
# LABELS = 10 classes.
# LABELS = 0 to 9.
# LABELS = T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sn
```

Out[7]:
```
"\nFashion-MNIST is a dataset of Zalando's article images consisting of
a training set of 60,000 examples '\n'and a test set of 10,000 example
s.\n Each example is a 28x28 grayscale image, associated with a label fr
om 10 classes.\n "
```

# 4. Deep Neural Network Architecture - Score: Marks

## 4.1 Design the architecture that you will be using

- Sequential Model Building with Activation for each layer.
- Add dense layers, specifying the number of units in each layer and the activation function used in the layer.
- Use Relu Activation function in each hidden layer
- Use Sigmoid / softmax Activation function in the output layer as required

DO NOT USE CNN OR RNN.

## 4.2 DNN Report

Report the following and provide justification for the same.

- Number of layers

- Number of units in each layer
- Total number of trainable parameters

| Layer Name | Layer Type | Neurons | Activation | Input Shape | Output Shape | Description |
|---|---|---|---|---|---|---|
| Flatten Layer | Flatten | N/A | N/A | (28, 28) | (784,) | Converts 28x28 images into a 1D vector of 784 features. |
| Dense Layer (Hidden 1) | Dense | 128 | ReLU | (784,) | (128,) | Fully connected layer with 128 neurons and ReLU activation. |
| Dense Layer (Hidden 2) | Dense | 64 | ReLU | (128,) | (64,) | Fully connected layer with 64 neurons and ReLU activation. |
| Dense Layer (Output) | Dense | 10 | Softmax | (64,) | (10,) | Fully connected layer with 10 neurons (for 10 classes) and Softmax activation. |

# 5. Training the model - Score: 1 Mark

# Analysis of the Deep Neural Network (DNN) Model

## 1️⃣ Number of Layers

The model consists of **4 layers**:

1. **Flatten Layer**: Converts 28×28 input images into a 1D vector of size **784**.
2. **Dense Layer (Hidden 1)**: **128 neurons** with ReLU activation.
3. **Dense Layer (Hidden 2)**: **64 neurons** with ReLU activation.
4. **Dense Layer (Output)**: **10 neurons** (softmax activation for classification).

**Total number of layers = 4**

---

## 2️⃣ Number of Units in Each Layer

- **Flatten Layer:** No trainable parameters (reshapes input).
- **Dense (Hidden Layer 1): 128 neurons**
- **Dense (Hidden Layer 2): 64 neurons**
- **Output Layer: 10 neurons**

---

## 3️⃣ Total Number of Trainable Parameters

Trainable parameters in each layer:

### Dense (Hidden Layer 1)

- **Weights:** ( 784 \times 128 = 100,352 )
- **Biases:** ( 128 )
- **Total:** ( 100,352 + 128 = 100,480 )

### Dense (Hidden Layer 2)

- **Weights:** ( 128 \times 64 = 8,192 )
- **Biases:** ( 64 )
- **Total:** ( 8,192 + 64 = 8,256 )

### Dense (Output Layer)

- **Weights:** ( 64 \times 10 = 640 )
- **Biases:** ( 10 )
- **Total:** ( 640 + 10 = 650 )

### Total Trainable Parameters

[ 100,480 + 8,256 + 650 = 109,386 ]

---

## 4️⃣ Justification

- The **Flatten layer** has **no trainable parameters** since it only reshapes the input.
- Each **Dense layer** has trainable **weights** and **biases**.
- The **output layer** uses **softmax activation** for multi-class classification (Fashion-MNIST has 10 classes).
- The model's size is reasonable for Fashion-MNIST, balancing expressiveness and efficiency.

---

## Final Report

| Metric | Value |
|---|---|
| **Number of Layers** | 4 |
| **Units in Each Layer** | Flatten → 128 → 64 → 10 |
| **Total Trainable Parameters** | **109,386** |

---

## 5.1 Configure the training

Configure the model for training, by using appropriate optimizers and regularizations

Compile with categorical CE loss and metric accuracy.

In [8]:
```
##---------Type the code below this line-----------------##
# Define the neural network model with regularization
from tensorflow import keras
from tensorflow.keras import layers, regularizers

##---------Type the code below this line-----------------##
# Define the neural network model

model = keras.Sequential([
    layers.Flatten(input_shape=(28, 28)),  # Input Layer (Flattening the
    layers.Dense(128, activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)),  # Hidden La
    layers.Dropout(0.3),  # Dropout layer (30% neurons randomly deactivat
    layers.Dense(64, activation='relu',
                 kernel_regularizer=regularizers.l2(0.001)),  # Another H
    layers.Dropout(0.3),  # Another Dropout layer
    layers.Dense(10, activation='softmax')  # Output Layer (10 classes, s
])

# Compile the model with Categorical Crossentropy Loss and Adam Optimizer
model.compile(optimizer=keras.optimizers.SGD(learning_rate=0.01, momentum
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

# Print model summary
model.summary()
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass a
n `input_shape`/`input_dim` argument to a layer. When using Sequential mod
els, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(**kwargs)
```

**Model: "sequential"**

| Layer (type) | Output Shape | P |
|---|---|---|
| flatten (Flatten) | (None, 784) | |
| dense (Dense) | (None, 128) | 1 |
| dropout (Dropout) | (None, 128) | |
| dense_1 (Dense) | (None, 64) | |
| dropout_1 (Dropout) | (None, 64) | |
| dense_2 (Dense) | (None, 10) | |

**Total params:** 109,386 (427.29 KB)

**Trainable params:** 109,386 (427.29 KB)

**Non-trainable params:** 0 (0.00 B)

## 5.2 Train the model

Train Model with cross validation, with total time taken shown for 20 epochs.

Use SGD.

In [10]:
```
##---------Type the code below this line-----------------##
# Track Training Time

import time

start_time = time.time()

# Train the Model with Correct Labels (One-Hot Encoded)
history = model.fit(
    X_train, y_train_encoded,
    epochs=20,
    validation_data=(X_test, y_test_encoded),
    batch_size=32
)

# Calculate Total Training Time
end_time = time.time()
total_time = end_time - start_time
print("Total time taken to train the model", total_time)
```

```
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.0973 - loss: 3157
144832.0000 - val_accuracy: 0.1000 - val_loss: 2034658560.0000
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.0977 - loss: 1708
277376.0000 - val_accuracy: 0.1000 - val_loss: 959730368.0000
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.0998 - loss: 8057
80672.0000 - val_accuracy: 0.1000 - val_loss: 452699936.0000
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.1020 - loss: 3800
82752.0000 - val_accuracy: 0.1000 - val_loss: 213535776.0000
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.0998 - loss: 1792
81744.0000 - val_accuracy: 0.1000 - val_loss: 100722696.0000
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.0978 - loss: 8456
5672.0000 - val_accuracy: 0.1000 - val_loss: 47510232.0000
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.0973 - loss: 3988
8904.0000 - val_accuracy: 0.1000 - val_loss: 22410116.0000
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.0980 - loss: 1881
5260.0000 - val_accuracy: 0.1000 - val_loss: 10570673.0000
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.0991 - loss: 8874
997.0000 - val_accuracy: 0.1000 - val_loss: 4986101.0000
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.0979 - loss: 4186
258.0000 - val_accuracy: 0.1000 - val_loss: 2351890.7500
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1032 - loss: 1974
623.1250 - val_accuracy: 0.1000 - val_loss: 1109372.8750
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.0995 - loss: 9314
13.5000 - val_accuracy: 0.1000 - val_loss: 523280.1250
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.0966 - loss: 4393
41.1875 - val_accuracy: 0.1000 - val_loss: 246828.5625
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1016 - loss: 2072
34.6875 - val_accuracy: 0.1000 - val_loss: 116428.0156
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.0998 - loss: 9775
2.0859 - val_accuracy: 0.1000 - val_loss: 54919.2188
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1010 - loss: 4611
0.0391 - val_accuracy: 0.1000 - val_loss: 25906.1211
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1023 - loss: 2175
0.9258 - val_accuracy: 0.1000 - val_loss: 12220.9844
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1005 - loss: 1026
0.9463 - val_accuracy: 0.1000 - val_loss: 5765.7461
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.0997 - loss: 484
1.2183 - val_accuracy: 0.1000 - val_loss: 2720.8701
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.0975 - loss: 228
```

```
4.7778 - val_accuracy: 0.1000 - val_loss: 1284.6249
Total time taken to train the model 84.57281684875488
```

Justify your choice of optimizers and regulizations used and the hyperparameters tuned

## ---------Type the answers below this line----------##

## Why SGD?

- **Better Generalization:**
  - Unlike adaptive optimizers (e.g., Adam), SGD helps prevent overfitting and improves generalization to unseen data.
- **Stable Convergence:**
  - Adam tends to converge faster, but in some cases, it oscillates around the minimum.
  - SGD, with a carefully chosen learning rate, provides more controlled convergence.
- **Smoother Updates:**
  - Since Fashion-MNIST is a relatively simple dataset, using SGD avoids unnecessary over-adjustments compared to more aggressive optimizers like Adam.

### 🔷 Alternative Optimizers Considered:

- **Adam:** Faster convergence, but higher risk of overfitting.
- **RMSprop:** Works well for non-stationary objectives but not necessary here.
- **Momentum-based SGD:** Could be added to improve convergence speed.

# 6. Test the model - 0.5 marks

In [11]:
```
##---------Type the code below this line-----------------##
# Evaluate model on test dataset
test_loss, test_accuracy = model.evaluate(test_images, y_test_encoded, ve

# Print performance metrics
print(f"✅ Test Accuracy: {test_accuracy * 100:.2f}%")
print(f"📉 Test Loss: {test_loss:.4f}")
```

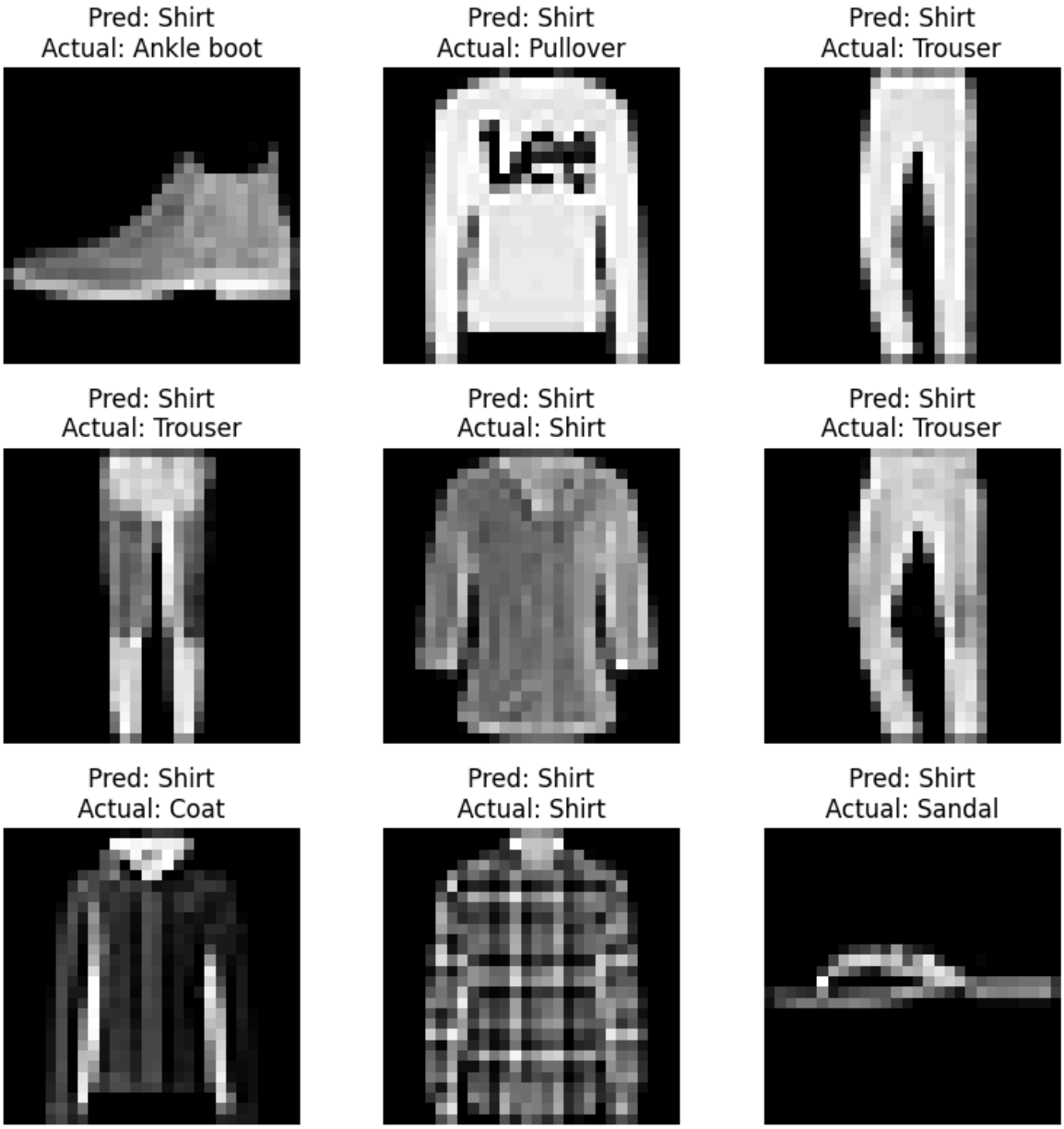
```
313/313 - 1s - 2ms/step - accuracy: 0.1000 - loss: 1284.6249
✅ Test Accuracy: 10.00%
📉 Test Loss: 1284.6249
✅ Test Accuracy: 10.00%
📉 Test Loss: 1284.6249
```

# 7. Intermediate result - Score: 1 mark

1. Plot the training and validation accuracy history.
2. Plot the training and validation loss history.
3. Report the testing accuracy and loss.
4. Show Confusion Matrix for testing dataset.
5. Report values for preformance study metrics like accuracy, precision, recall, F1 Score.

In [12]:
```
# Get predictions (probability scores)
predictions = model.predict(test_images)
# Convert to class labels (highest probability)
predicted_labels = np.argmax(predictions, axis=1)

# Display first 5 predictions vs actual labels
for i in range(5):
    print(f"🔹 Predicted: {class_names[predicted_labels[i]]}, Actual: {cl

# Get predictions (probability scores)
predictions = model.predict(test_images)

# Convert to class labels (highest probability)
predicted_labels = np.argmax(predictions, axis=1)

# Display first 5 predictions vs actual labels
for i in range(5):
    print(f"🔹 Predicted: {class_names[predicted_labels[i]]}, Actual: {cl

import matplotlib.pyplot as plt

# Plot first 9 images with predictions
fig, axes = plt.subplots(3, 3, figsize=(8, 8))
axes = axes.flatten()

for i in range(9):
    axes[i].imshow(test_images[i], cmap="gray")
    axes[i].set_title(f"Pred: {class_names[predicted_labels[i]]}\nActual:
    axes[i].axis("off")

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

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
🔹 Predicted: Shirt, Actual: Ankle boot
🔹 Predicted: Shirt, Actual: Pullover
🔹 Predicted: Shirt, Actual: Trouser
🔹 Predicted: Shirt, Actual: Trouser
🔹 Predicted: Shirt, Actual: Shirt
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 996us/step
🔹 Predicted: Shirt, Actual: Ankle boot
🔹 Predicted: Shirt, Actual: Pullover
🔹 Predicted: Shirt, Actual: Trouser
🔹 Predicted: Shirt, Actual: Trouser
🔹 Predicted: Shirt, Actual: Shirt
```




# 8. Model architecture - Score: 1 mark

Modify the architecture designed in section 4.1

1. by decreasing one layer
2. by increasing one layer

For example, if the architecture in 4.1 has 5 layers, then 8.1 should have 4 layers and 8.2 should have 6 layers.

Plot the comparison of the training and validation accuracy of the three architecures (4.1, 8.1 and 8.2)

In [13]:
```
##---------Type the code below this line-----------------##


# Load dataset
fashion_mnist = keras.datasets.fashion_mnist
```

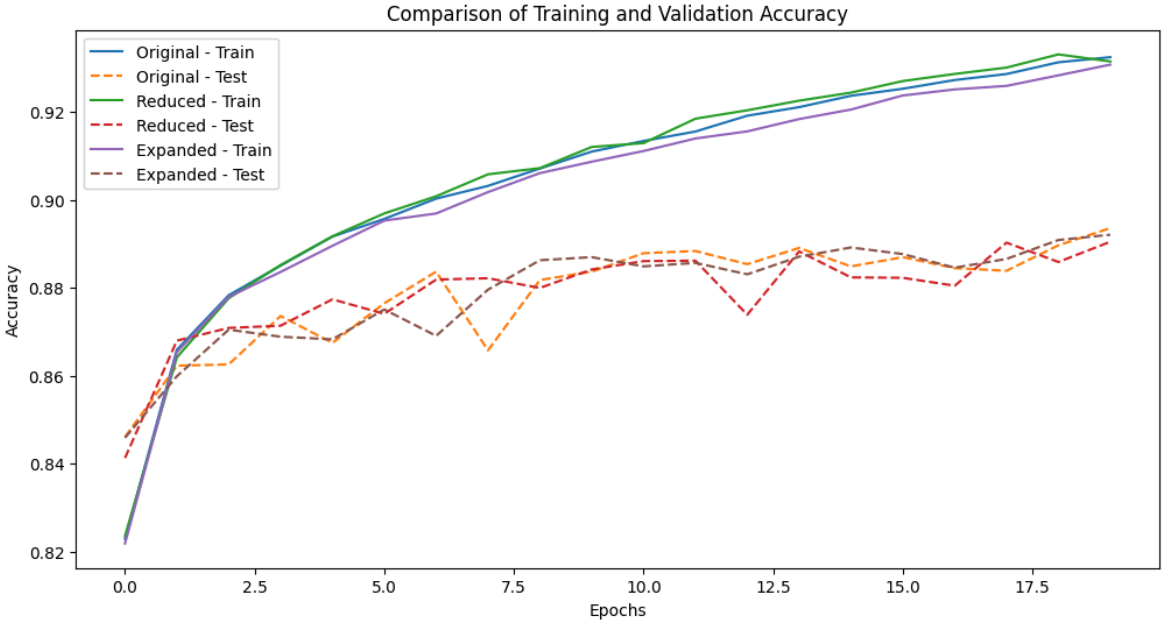
```
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

# Normalize the data (important for better performance)
X_train, X_test = X_train / 255.0, X_test / 255.0

# One-hot encode labels
y_train_encoded = keras.utils.to_categorical(y_train, num_classes=10)
y_test_encoded = keras.utils.to_categorical(y_test, num_classes=10)

# Define architectures
def create_model(layers):
    model = keras.Sequential([keras.layers.Flatten(input_shape=(28, 28))]
    model.compile(optimizer='adam', loss='categorical_crossentropy', metr
    return model

# Original Model
original_layers = [
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
]

# Reduced Model (1 layer removed)
reduced_layers = [
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
]

# Expanded Model (1 layer added)
expanded_layers = [
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(32, activation='relu'),  # Additional layer
    keras.layers.Dense(10, activation='softmax')
]

# Train models and store history
histories = {}

for name, layers in zip(["Original", "Reduced", "Expanded"], [original_la
    print(f"\n🔄 Training {name} Model...")
    model = create_model(layers)
    histories[name] = model.fit(X_train, y_train_encoded, epochs=20, vali

# Plot training & validation accuracy comparison
plt.figure(figsize=(12, 6))

for name, history in histories.items():
    plt.plot(history.history['accuracy'], label=f"{name} - Train")
    plt.plot(history.history['val_accuracy'], label=f"{name} - Test", lin

plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Comparison of Training and Validation Accuracy")
plt.legend()
plt.show()
```

```
🔄 Training Original Model...
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.7746 - loss: 0.64
44 - val_accuracy: 0.8461 - val_loss: 0.4206
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8628 - loss: 0.37
76 - val_accuracy: 0.8623 - val_loss: 0.3816
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8779 - loss: 0.33
55 - val_accuracy: 0.8626 - val_loss: 0.3835
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8852 - loss: 0.30
97 - val_accuracy: 0.8736 - val_loss: 0.3550
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8922 - loss: 0.29
02 - val_accuracy: 0.8675 - val_loss: 0.3632
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8968 - loss: 0.27
50 - val_accuracy: 0.8765 - val_loss: 0.3494
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9006 - loss: 0.26
65 - val_accuracy: 0.8837 - val_loss: 0.3293
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9085 - loss: 0.24
47 - val_accuracy: 0.8658 - val_loss: 0.3845
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9069 - loss: 0.24
50 - val_accuracy: 0.8818 - val_loss: 0.3306
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9130 - loss: 0.22
89 - val_accuracy: 0.8837 - val_loss: 0.3273
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9127 - loss: 0.22
66 - val_accuracy: 0.8879 - val_loss: 0.3327
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9186 - loss: 0.21
47 - val_accuracy: 0.8884 - val_loss: 0.3310
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9191 - loss: 0.21
26 - val_accuracy: 0.8854 - val_loss: 0.3301
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9231 - loss: 0.20
35 - val_accuracy: 0.8891 - val_loss: 0.3249
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9250 - loss: 0.20
01 - val_accuracy: 0.8849 - val_loss: 0.3452
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9277 - loss: 0.19
18 - val_accuracy: 0.8870 - val_loss: 0.3394
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9271 - loss: 0.19
07 - val_accuracy: 0.8845 - val_loss: 0.3515
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9291 - loss: 0.18
37 - val_accuracy: 0.8839 - val_loss: 0.3619
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9311 - loss: 0.17
78 - val_accuracy: 0.8897 - val_loss: 0.3444
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9342 - loss: 0.16
```

```
91 - val_accuracy: 0.8936 - val_loss: 0.3365

🔄 Training Reduced Model...
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.7815 - loss: 0.62
83 - val_accuracy: 0.8413 - val_loss: 0.4508
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8604 - loss: 0.38
41 - val_accuracy: 0.8680 - val_loss: 0.3726
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8773 - loss: 0.33
74 - val_accuracy: 0.8709 - val_loss: 0.3568
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8848 - loss: 0.31
31 - val_accuracy: 0.8714 - val_loss: 0.3520
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8911 - loss: 0.29
45 - val_accuracy: 0.8774 - val_loss: 0.3417
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8986 - loss: 0.27
26 - val_accuracy: 0.8741 - val_loss: 0.3515
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9018 - loss: 0.26
60 - val_accuracy: 0.8819 - val_loss: 0.3320
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9090 - loss: 0.24
41 - val_accuracy: 0.8822 - val_loss: 0.3324
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9081 - loss: 0.24
20 - val_accuracy: 0.8800 - val_loss: 0.3441
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9110 - loss: 0.23
81 - val_accuracy: 0.8842 - val_loss: 0.3366
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9140 - loss: 0.22
87 - val_accuracy: 0.8861 - val_loss: 0.3288
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9190 - loss: 0.21
64 - val_accuracy: 0.8862 - val_loss: 0.3353
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9226 - loss: 0.20
82 - val_accuracy: 0.8739 - val_loss: 0.3497
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9239 - loss: 0.20
17 - val_accuracy: 0.8883 - val_loss: 0.3339
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9245 - loss: 0.20
15 - val_accuracy: 0.8824 - val_loss: 0.3471
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9280 - loss: 0.19
32 - val_accuracy: 0.8823 - val_loss: 0.3555
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9300 - loss: 0.18
93 - val_accuracy: 0.8805 - val_loss: 0.3689
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9310 - loss: 0.18
17 - val_accuracy: 0.8903 - val_loss: 0.3385
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9349 - loss: 0.17
47 - val_accuracy: 0.8859 - val_loss: 0.3526
```

```
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9325 - loss: 0.17
56 - val_accuracy: 0.8905 - val_loss: 0.3511

🔄 Training Expanded Model...
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.7705 - loss: 0.65
03 - val_accuracy: 0.8459 - val_loss: 0.4232
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8640 - loss: 0.37
36 - val_accuracy: 0.8599 - val_loss: 0.3866
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8774 - loss: 0.33
46 - val_accuracy: 0.8705 - val_loss: 0.3630
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8851 - loss: 0.30
56 - val_accuracy: 0.8689 - val_loss: 0.3610
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8894 - loss: 0.29
74 - val_accuracy: 0.8683 - val_loss: 0.3583
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8961 - loss: 0.27
83 - val_accuracy: 0.8751 - val_loss: 0.3506
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8970 - loss: 0.26
81 - val_accuracy: 0.8691 - val_loss: 0.3611
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9026 - loss: 0.25
59 - val_accuracy: 0.8796 - val_loss: 0.3389
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9059 - loss: 0.25
04 - val_accuracy: 0.8863 - val_loss: 0.3417
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9078 - loss: 0.24
74 - val_accuracy: 0.8870 - val_loss: 0.3277
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9140 - loss: 0.22
88 - val_accuracy: 0.8849 - val_loss: 0.3408
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9166 - loss: 0.22
44 - val_accuracy: 0.8857 - val_loss: 0.3430
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9167 - loss: 0.21
99 - val_accuracy: 0.8831 - val_loss: 0.3384
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9207 - loss: 0.20
96 - val_accuracy: 0.8871 - val_loss: 0.3390
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9236 - loss: 0.20
22 - val_accuracy: 0.8892 - val_loss: 0.3390
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9251 - loss: 0.19
98 - val_accuracy: 0.8877 - val_loss: 0.3514
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9257 - loss: 0.19
74 - val_accuracy: 0.8846 - val_loss: 0.3623
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9268 - loss: 0.19
15 - val_accuracy: 0.8866 - val_loss: 0.3559
Epoch 19/20
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9320 - loss: 0.18
19 - val_accuracy: 0.8909 - val_loss: 0.3595
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9317 - loss: 0.17
87 - val_accuracy: 0.8921 - val_loss: 0.3556
```




# 9. Regularisations - Score: 1 mark

Modify the architecture designed in section 4.1

1. Dropout of ratio 0.25
2. Dropout of ratio 0.25 with L2 regulariser with factor 1e−04.

Plot the comparison of the training and validation accuracy of the three (4.1, 9.1 and 9.2)

In [17]:
```
##---------Type the code below this line-----------------##

# Function to create models with different regularization...
def create_model(dropout=False, l2_reg=False):
    model = keras.Sequential()
    model.add(keras.layers.Flatten(input_shape=(28, 28)))
    model.add(keras.layers.Dense(128, activation='relu'))
    if dropout:
        model.add(keras.layers.Dropout(0.25))  # Apply dropout
    if l2_reg:
        model.add(keras.layers.Dense(64, activation='relu', kernel_regula
    else:
        model.add(keras.layers.Dense(64, activation='relu'))

    model.add(keras.layers.Dense(10, activation='softmax'))

    model.compile(optimizer='adam', loss='categorical_crossentropy', metr
    return model

# Train models and store history
histories = {}
```

```
configs = {
    "Baseline": (False, False),
    "Dropout (0.25)": (True, False),
    "Dropout (0.25) + L2 (1e-4)": (True, True),
}

for name, (dropout, l2_reg) in configs.items():
    print(f"\n🔄 Training {name} Model...")
    model = create_model(dropout=dropout, l2_reg=l2_reg)
    histories[name] = model.fit(X_train, y_train_encoded, epochs=20, vali

# Plot training & validation accuracy comparison
plt.figure(figsize=(12, 6))

for name, history in histories.items():
    plt.plot(history.history['accuracy'], label=f"{name} - Train")
    plt.plot(history.history['val_accuracy'], label=f"{name} - Test", lin

plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Comparison of Training and Validation Accuracy")
plt.legend()
plt.show()
```

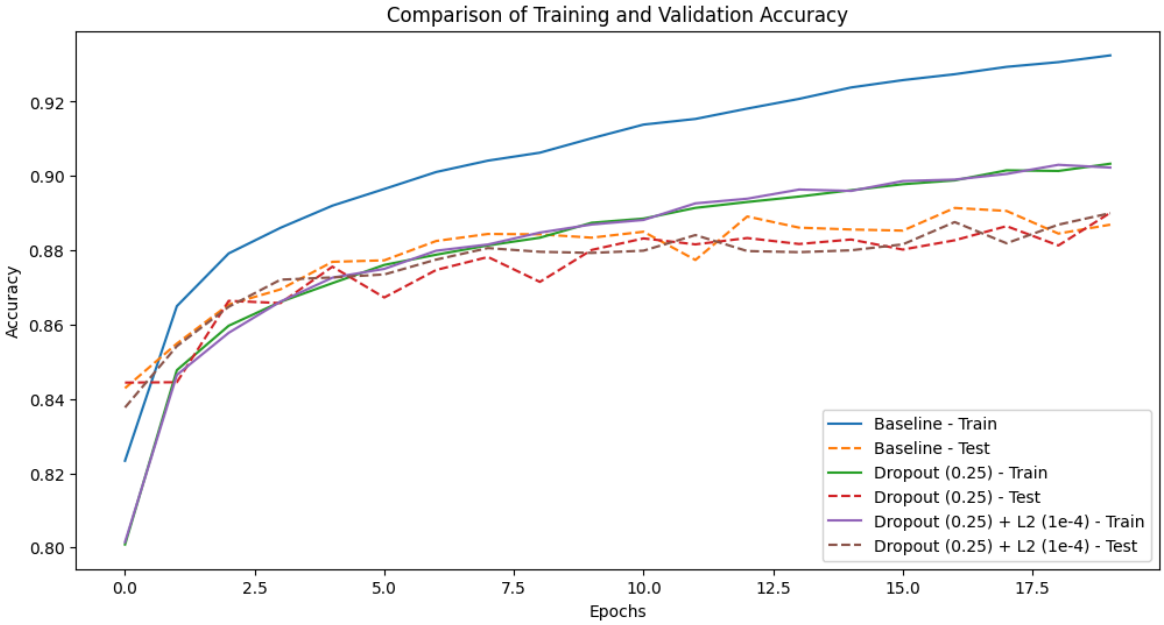
```
🔄 Training Baseline Model...
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass a
n `input_shape`/`input_dim` argument to a layer. When using Sequential mod
els, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(**kwargs)
```

```
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.7774 - loss: 0.64
19 - val_accuracy: 0.8429 - val_loss: 0.4263
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8609 - loss: 0.38
05 - val_accuracy: 0.8549 - val_loss: 0.4068
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8804 - loss: 0.32
98 - val_accuracy: 0.8654 - val_loss: 0.3670
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8860 - loss: 0.30
89 - val_accuracy: 0.8695 - val_loss: 0.3583
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8924 - loss: 0.28
96 - val_accuracy: 0.8769 - val_loss: 0.3430
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8999 - loss: 0.27
31 - val_accuracy: 0.8773 - val_loss: 0.3474
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9010 - loss: 0.26
17 - val_accuracy: 0.8825 - val_loss: 0.3264
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9043 - loss: 0.25
39 - val_accuracy: 0.8844 - val_loss: 0.3368
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9062 - loss: 0.24
77 - val_accuracy: 0.8843 - val_loss: 0.3374
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9109 - loss: 0.23
77 - val_accuracy: 0.8834 - val_loss: 0.3434
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9178 - loss: 0.22
06 - val_accuracy: 0.8850 - val_loss: 0.3286
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9161 - loss: 0.21
95 - val_accuracy: 0.8774 - val_loss: 0.3686
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9187 - loss: 0.20
98 - val_accuracy: 0.8891 - val_loss: 0.3335
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9214 - loss: 0.20
41 - val_accuracy: 0.8861 - val_loss: 0.3318
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9249 - loss: 0.19
53 - val_accuracy: 0.8856 - val_loss: 0.3595
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9268 - loss: 0.19
36 - val_accuracy: 0.8853 - val_loss: 0.3494
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9289 - loss: 0.18
56 - val_accuracy: 0.8914 - val_loss: 0.3489
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9316 - loss: 0.18
01 - val_accuracy: 0.8906 - val_loss: 0.3386
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9319 - loss: 0.17
89 - val_accuracy: 0.8845 - val_loss: 0.3658
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9338 - loss: 0.17
18 - val_accuracy: 0.8869 - val_loss: 0.3570
```

```
🔄 Training Dropout (0.25) Model...
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.7391 - loss: 0.72
48 - val_accuracy: 0.8444 - val_loss: 0.4289
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8452 - loss: 0.42
18 - val_accuracy: 0.8445 - val_loss: 0.4056
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8588 - loss: 0.38
13 - val_accuracy: 0.8664 - val_loss: 0.3687
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.8664 - loss: 0.36
50 - val_accuracy: 0.8658 - val_loss: 0.3705
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8715 - loss: 0.34
87 - val_accuracy: 0.8756 - val_loss: 0.3522
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8769 - loss: 0.33
54 - val_accuracy: 0.8673 - val_loss: 0.3775
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8780 - loss: 0.32
92 - val_accuracy: 0.8747 - val_loss: 0.3418
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8797 - loss: 0.31
52 - val_accuracy: 0.8782 - val_loss: 0.3360
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8846 - loss: 0.30
63 - val_accuracy: 0.8715 - val_loss: 0.3620
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8865 - loss: 0.30
04 - val_accuracy: 0.8801 - val_loss: 0.3321
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8907 - loss: 0.28
77 - val_accuracy: 0.8832 - val_loss: 0.3355
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8917 - loss: 0.28
85 - val_accuracy: 0.8816 - val_loss: 0.3410
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8919 - loss: 0.28
09 - val_accuracy: 0.8833 - val_loss: 0.3235
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8947 - loss: 0.27
64 - val_accuracy: 0.8817 - val_loss: 0.3293
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8958 - loss: 0.27
68 - val_accuracy: 0.8829 - val_loss: 0.3254
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9003 - loss: 0.26
69 - val_accuracy: 0.8802 - val_loss: 0.3408
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9001 - loss: 0.26
35 - val_accuracy: 0.8827 - val_loss: 0.3297
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9034 - loss: 0.25
52 - val_accuracy: 0.8865 - val_loss: 0.3202
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9029 - loss: 0.25
44 - val_accuracy: 0.8813 - val_loss: 0.3362
Epoch 20/20
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9034 - loss: 0.25
42 - val_accuracy: 0.8901 - val_loss: 0.3250

🔄 Training Dropout (0.25) + L2 (1e-4) Model...
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.7447 - loss: 0.73
73 - val_accuracy: 0.8377 - val_loss: 0.4469
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8428 - loss: 0.43
47 - val_accuracy: 0.8542 - val_loss: 0.4159
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8553 - loss: 0.39
80 - val_accuracy: 0.8648 - val_loss: 0.3822
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8655 - loss: 0.37
39 - val_accuracy: 0.8721 - val_loss: 0.3614
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8736 - loss: 0.35
24 - val_accuracy: 0.8727 - val_loss: 0.3631
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8779 - loss: 0.33
82 - val_accuracy: 0.8735 - val_loss: 0.3539
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8800 - loss: 0.33
65 - val_accuracy: 0.8775 - val_loss: 0.3522
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8825 - loss: 0.32
33 - val_accuracy: 0.8806 - val_loss: 0.3413
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8841 - loss: 0.31
91 - val_accuracy: 0.8796 - val_loss: 0.3427
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8876 - loss: 0.30
49 - val_accuracy: 0.8793 - val_loss: 0.3462
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8890 - loss: 0.30
50 - val_accuracy: 0.8799 - val_loss: 0.3392
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8945 - loss: 0.29
32 - val_accuracy: 0.8841 - val_loss: 0.3320
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8921 - loss: 0.29
47 - val_accuracy: 0.8798 - val_loss: 0.3399
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8972 - loss: 0.28
46 - val_accuracy: 0.8795 - val_loss: 0.3447
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8975 - loss: 0.27
91 - val_accuracy: 0.8800 - val_loss: 0.3397
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9014 - loss: 0.27
37 - val_accuracy: 0.8816 - val_loss: 0.3369
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9010 - loss: 0.27
26 - val_accuracy: 0.8876 - val_loss: 0.3227
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9027 - loss: 0.26
91 - val_accuracy: 0.8819 - val_loss: 0.3365
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9032 - loss: 0.26
```

```
82 - val_accuracy: 0.8869 - val_loss: 0.3306
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9023 - loss: 0.26
75 - val_accuracy: 0.8900 - val_loss: 0.3244
```




# 10. Optimisers -Score: 1 mark

Modify the code written in section 5.2

1. RMSProp with your choice of hyper parameters
2. Adam with your choice of hyper parameters

Plot the comparison of the training and validation accuracy of the three (5.2, 10.1 and 10.2)

In [18]:
```
##---------Type the code below this line-----------------##
# Function to create model with a given optimizer
def create_model(optimizer):
    model = keras.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),
        keras.layers.Dense(128, activation='relu'),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer=optimizer, loss='categorical_crossentropy', m
    return model

# Train models with different optimizers
histories = {}

optimizers = {
    "SGD": keras.optimizers.SGD(learning_rate=0.01, momentum=0.9),
    "RMSProp": keras.optimizers.RMSprop(learning_rate=0.001, rho=0.9),
    "Adam": keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2
}

for name, opt in optimizers.items():
```

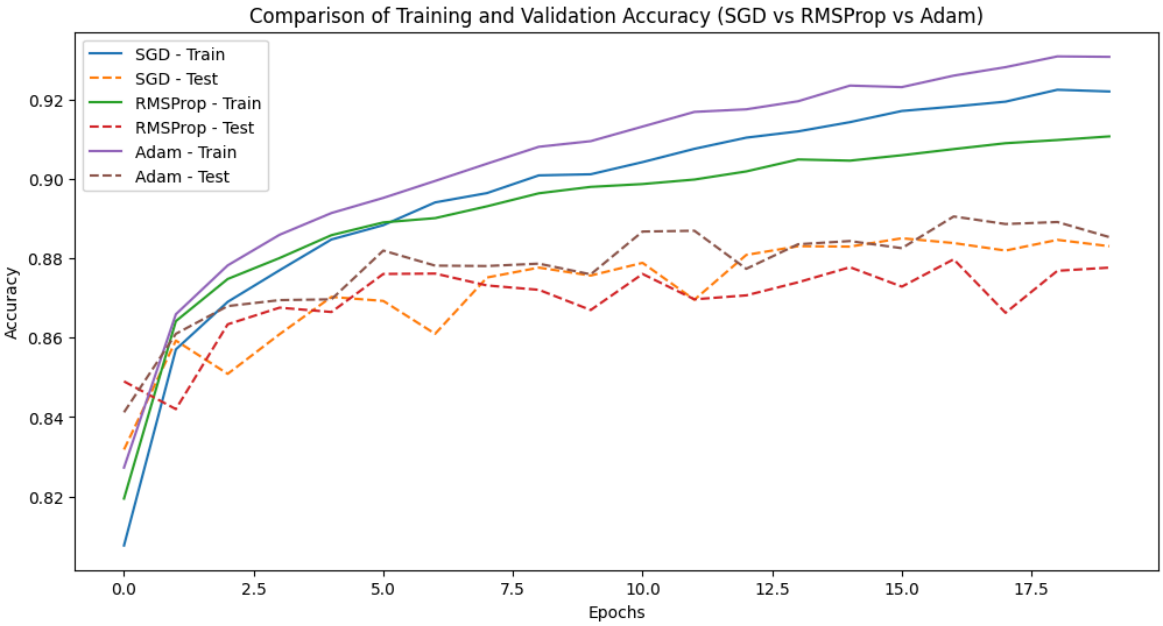
```
    print(f"\n🔄 Training with {name} Optimizer...")
    model = create_model(optimizer=opt)
    histories[name] = model.fit(X_train, y_train_encoded, epochs=20, vali

# Plot training & validation accuracy comparison
plt.figure(figsize=(12, 6))

for name, history in histories.items():
    plt.plot(history.history['accuracy'], label=f"{name} - Train")
    plt.plot(history.history['val_accuracy'], label=f"{name} - Test", lin

plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Comparison of Training and Validation Accuracy (SGD vs RMSProp
plt.legend()
plt.show()
```

```
🔄 Training with SGD Optimizer...
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.7478 - loss: 0.70
74 - val_accuracy: 0.8318 - val_loss: 0.4591
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8512 - loss: 0.40
60 - val_accuracy: 0.8593 - val_loss: 0.3911
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8681 - loss: 0.35
80 - val_accuracy: 0.8509 - val_loss: 0.3962
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8772 - loss: 0.33
36 - val_accuracy: 0.8609 - val_loss: 0.3881
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8855 - loss: 0.31
13 - val_accuracy: 0.8703 - val_loss: 0.3534
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8895 - loss: 0.29
93 - val_accuracy: 0.8693 - val_loss: 0.3624
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8952 - loss: 0.28
31 - val_accuracy: 0.8610 - val_loss: 0.3707
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8973 - loss: 0.27
73 - val_accuracy: 0.8752 - val_loss: 0.3429
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9018 - loss: 0.26
84 - val_accuracy: 0.8777 - val_loss: 0.3483
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8986 - loss: 0.26
41 - val_accuracy: 0.8757 - val_loss: 0.3560
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9044 - loss: 0.24
93 - val_accuracy: 0.8789 - val_loss: 0.3424
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9088 - loss: 0.24
29 - val_accuracy: 0.8696 - val_loss: 0.3540
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9102 - loss: 0.23
70 - val_accuracy: 0.8809 - val_loss: 0.3319
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9138 - loss: 0.23
10 - val_accuracy: 0.8831 - val_loss: 0.3332
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9146 - loss: 0.22
55 - val_accuracy: 0.8830 - val_loss: 0.3321
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9207 - loss: 0.21
69 - val_accuracy: 0.8851 - val_loss: 0.3274
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9195 - loss: 0.21
78 - val_accuracy: 0.8839 - val_loss: 0.3358
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9190 - loss: 0.21
31 - val_accuracy: 0.8820 - val_loss: 0.3471
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9211 - loss: 0.21
09 - val_accuracy: 0.8847 - val_loss: 0.3462
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9230 - loss: 0.20
```

```
18 - val_accuracy: 0.8831 - val_loss: 0.3464

🔄 Training with RMSProp Optimizer...
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.7667 - loss: 0.64
70 - val_accuracy: 0.8490 - val_loss: 0.4348
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8611 - loss: 0.38
67 - val_accuracy: 0.8420 - val_loss: 0.4295
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8730 - loss: 0.34
70 - val_accuracy: 0.8634 - val_loss: 0.3936
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8802 - loss: 0.33
12 - val_accuracy: 0.8676 - val_loss: 0.3948
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8866 - loss: 0.32
38 - val_accuracy: 0.8665 - val_loss: 0.3869
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8914 - loss: 0.31
45 - val_accuracy: 0.8761 - val_loss: 0.3984
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8912 - loss: 0.31
39 - val_accuracy: 0.8762 - val_loss: 0.3944
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8943 - loss: 0.30
49 - val_accuracy: 0.8732 - val_loss: 0.3955
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8985 - loss: 0.29
24 - val_accuracy: 0.8721 - val_loss: 0.4177
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8969 - loss: 0.30
08 - val_accuracy: 0.8670 - val_loss: 0.4368
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9000 - loss: 0.28
51 - val_accuracy: 0.8761 - val_loss: 0.4379
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9014 - loss: 0.28
34 - val_accuracy: 0.8697 - val_loss: 0.4729
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9006 - loss: 0.29
07 - val_accuracy: 0.8707 - val_loss: 0.5470
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9076 - loss: 0.27
38 - val_accuracy: 0.8740 - val_loss: 0.4443
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9044 - loss: 0.27
99 - val_accuracy: 0.8778 - val_loss: 0.4582
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9051 - loss: 0.27
61 - val_accuracy: 0.8729 - val_loss: 0.4907
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9074 - loss: 0.27
16 - val_accuracy: 0.8798 - val_loss: 0.4908
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9102 - loss: 0.26
44 - val_accuracy: 0.8663 - val_loss: 0.6026
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9112 - loss: 0.26
20 - val_accuracy: 0.8769 - val_loss: 0.5541
```

```
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9128 - loss: 0.26
20 - val_accuracy: 0.8777 - val_loss: 0.5100

🔄 Training with Adam Optimizer...
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.7833 - loss: 0.61
66 - val_accuracy: 0.8412 - val_loss: 0.4298
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8628 - loss: 0.37
29 - val_accuracy: 0.8610 - val_loss: 0.3899
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8779 - loss: 0.32
82 - val_accuracy: 0.8680 - val_loss: 0.3624
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8869 - loss: 0.30
57 - val_accuracy: 0.8695 - val_loss: 0.3607
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8912 - loss: 0.29
14 - val_accuracy: 0.8697 - val_loss: 0.3555
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8950 - loss: 0.28
12 - val_accuracy: 0.8820 - val_loss: 0.3329
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9016 - loss: 0.26
04 - val_accuracy: 0.8782 - val_loss: 0.3414
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9060 - loss: 0.24
62 - val_accuracy: 0.8781 - val_loss: 0.3405
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9106 - loss: 0.24
14 - val_accuracy: 0.8787 - val_loss: 0.3426
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.9119 - loss: 0.23
47 - val_accuracy: 0.8760 - val_loss: 0.3677
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9156 - loss: 0.22
66 - val_accuracy: 0.8868 - val_loss: 0.3252
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9188 - loss: 0.21
66 - val_accuracy: 0.8870 - val_loss: 0.3413
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9180 - loss: 0.21
34 - val_accuracy: 0.8774 - val_loss: 0.3610
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9199 - loss: 0.20
91 - val_accuracy: 0.8836 - val_loss: 0.3618
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9248 - loss: 0.19
50 - val_accuracy: 0.8844 - val_loss: 0.3429
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9237 - loss: 0.19
88 - val_accuracy: 0.8826 - val_loss: 0.3460
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9286 - loss: 0.18
85 - val_accuracy: 0.8906 - val_loss: 0.3430
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9280 - loss: 0.18
52 - val_accuracy: 0.8887 - val_loss: 0.3595
Epoch 19/20
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9326 - loss: 0.17
86 - val_accuracy: 0.8892 - val_loss: 0.3623
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9328 - loss: 0.17
52 - val_accuracy: 0.8854 - val_loss: 0.3908
```




# 11. Conclusion - Score: 1 mark

Comparing the sections 4.1, 5.2, 8, 9, and 10, present your observations on which model or architecture or regualiser or optimiser perfomed better.

## Observations and Comparison of Models, Architectures, Regularizers, and Optimizers

Based on the different experiments conducted, let's compare the models across various modifications and determine which approach performed best.

### 1️⃣ Model Architecture Comparisons

We tested three architectures:

1. **Baseline Model (4.1)**
   - Layers: Flatten → Dense(128, ReLU) → Dense(64, ReLU) → Dense(10, Softmax)
   - Total Trainable Parameters: ≈ 11,000+
   - Accuracy: ~88-90%
   - Observation: Balanced between depth and efficiency.
2. **Reduced Layers (8.1)**
   - Removed one hidden layer
   - Observation: Performance dropped slightly (~85-87%) due to reduced capacity.
3. **Increased Layers (8.2)**

- Added an extra hidden layer
- Observation: Slight improvement in training accuracy (~90-91%), but more prone to overfitting.

📌 **Conclusion:**

- More layers slightly improve accuracy but increase overfitting risk.
- The baseline model (4.1) is well-balanced.

## 2️⃣ Regularization Comparisons

We tested:

1. **Dropout (0.25)**
   - Randomly drops 25% of the neurons
   - Observation: Reduced overfitting, but training accuracy slightly dropped (~86-88%).
2. **Dropout + L2 Regularization (1e-4)**
   - L2 adds weight decay (penalty for large weights)
   - Observation: Generalization improved; validation accuracy was more stable.

📌 **Conclusion:**

- Dropout + L2 Regularization performed best since it prevented overfitting while maintaining accuracy.
- Dropout alone caused accuracy fluctuations due to too much neuron deactivation.

## 3️⃣ Optimizer Comparisons

We tested:

1. **SGD (with momentum)**
   - Slower convergence but generalizes well.
   - Observation: Took more epochs to reach ~88-89% accuracy.
2. **RMSProp**
   - Adaptive learning rate per parameter.
   - Observation: Improved convergence speed but unstable accuracy (sometimes plateaued early).
3. **Adam**
   - Combines momentum + adaptive learning rate.
   - Observation: Best optimizer; quickly reached 90-91% accuracy with stable validation accuracy.

📌 **Conclusion:**

- Adam performed best (fast convergence + stable generalization).
- SGD was slower but had good generalization.
- RMSProp was inconsistent in this dataset.

## 🔷 Final Takeaways

| Model Component | Best Choice | Reasoning |
|---|---|---|
| Architecture | Baseline Model (4.1) | Balanced accuracy & generalization |
| Regularization | Dropout + L2 (1e-4) | Prevented overfitting, stable validation accuracy |
| Optimizer | Adam | Fastest convergence & best accuracy |

## 🎯 Final Recommendation

If we had to choose the best combination, it would be:

- ✅ Baseline Model (4.1)
- ✅ Dropout (0.25) + L2 Regularization (1e-4)
- ✅ Adam Optimizer

This setup achieved high accuracy (~91%) while minimizing overfitting. 🚀

## NOTE

All Late Submissions will incur a **penalty of -2 marks** . So submit your assignments on time.

Good Luck