Loading CIFAR-10

In [ ]:
```
import tensorflow as tf
from tensorflow.keras import layers, models, datasets
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from tensorflow.keras.regularizers import l2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
# Load and preprocess the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data(
train_images = train_images / 255.0
test_images = test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
```

Model Building

In [ ]:
```
# Define models
MLP_BN_model = models.Sequential([
    layers.Flatten(input_shape=(32, 32, 3)),
    layers.Dense(512),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Dropout(0.2),
    layers.Dense(256),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Dropout(0.2),
    layers.Dense(10, activation='softmax')
])
```

In [ ]:
```
MLP_L2_model = models.Sequential([
    layers.Flatten(input_shape=(32, 32, 3)),
    layers.Dense(512, kernel_regularizer=l2(0.001), activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(10, activation='softmax')
])
```

In [ ]:
```
Simple_CNN_model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

In [ ]:
```
CNN_DA_Dropout_model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
```

```
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
```

In [ ]:
```
ResNet_model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.BatchNormalization(),
    layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
```

Model Training

In [ ]:
```
# Compile all models
models = [MLP_BN_model, MLP_L2_model, Simple_CNN_model, CNN_DA_Dropout_model, ResNet_m
model_names = ["MLP with BatchNorm", "MLP with L2", "Simple CNN", "CNN with DA and Dro

i=0
for model in models:
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['
    print(model_names[i]+" Model Architecture : \n")
    print(model.summary())
    i+=1

# Train all models and store results
results = []

for idx, (model, name) in enumerate(zip(models, model_names), start=1):
    print(f"Training Model {idx}: {name}...")
    history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_i

    # Evaluate the model
    test_loss, test_acc = model.evaluate(test_images, test_labels)
    print(f'Test accuracy for {name}:', test_acc)
    results.append((history.history, test_loss, test_acc))
```

```
MLP with BatchNorm Model Architecture :

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 3072)              0

 dense (Dense)               (None, 512)               1573376

 batch_normalization (Batch  (None, 512)               2048
 Normalization)

 activation (Activation)     (None, 512)               0

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 batch_normalization_1 (Bat  (None, 256)               1024
 chNormalization)

 activation_1 (Activation)   (None, 256)               0

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 10)                2570

=================================================================
Total params: 1710346 (6.52 MB)
Trainable params: 1708810 (6.52 MB)
Non-trainable params: 1536 (6.00 KB)
_________________________________________________________________
None
MLP with L2 Model Architecture :

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 3072)              0

 dense_3 (Dense)             (None, 512)               1573376

 dropout_2 (Dropout)         (None, 512)               0

 dense_4 (Dense)             (None, 10)                5130

=================================================================
Total params: 1578506 (6.02 MB)
Trainable params: 1578506 (6.02 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Simple CNN Model Architecture :

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 flatten_2 (Flatten)         (None, 2304)              0

 dense_5 (Dense)             (None, 64)                147520

 dense_6 (Dense)             (None, 10)                650

=================================================================
Total params: 167562 (654.54 KB)
Trainable params: 167562 (654.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
CNN with DA and Dropout Model Architecture :

Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_2 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 flatten_3 (Flatten)         (None, 2304)              0

 dense_7 (Dense)             (None, 128)               295040

 dropout_3 (Dropout)         (None, 128)               0

 dense_8 (Dense)             (None, 10)                1290

=================================================================
Total params: 315722 (1.20 MB)
Trainable params: 315722 (1.20 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
ResNet Model Architecture :

Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 30, 30, 32)        896
```

```
 batch_normalization_2 (Bat  (None, 30, 30, 32)        128
 chNormalization)

 conv2d_5 (Conv2D)           (None, 30, 30, 32)        9248

 batch_normalization_3 (Bat  (None, 30, 30, 32)        128
 chNormalization)

 max_pooling2d_4 (MaxPoolin  (None, 15, 15, 32)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 15, 15, 64)        18496

 batch_normalization_4 (Bat  (None, 15, 15, 64)        256
 chNormalization)

 conv2d_7 (Conv2D)           (None, 15, 15, 64)        36928

 batch_normalization_5 (Bat  (None, 15, 15, 64)        256
 chNormalization)

 max_pooling2d_5 (MaxPoolin  (None, 7, 7, 64)          0
 g2D)

 conv2d_8 (Conv2D)           (None, 7, 7, 128)         73856

 batch_normalization_6 (Bat  (None, 7, 7, 128)         512
 chNormalization)

 conv2d_9 (Conv2D)           (None, 7, 7, 128)         147584

 batch_normalization_7 (Bat  (None, 7, 7, 128)         512
 chNormalization)

 max_pooling2d_6 (MaxPoolin  (None, 3, 3, 128)         0
 g2D)

 flatten_4 (Flatten)         (None, 1152)              0

 dense_9 (Dense)             (None, 128)               147584

 dropout_4 (Dropout)         (None, 128)               0

 dense_10 (Dense)            (None, 10)                1290

=================================================================
Total params: 437674 (1.67 MB)
Trainable params: 436778 (1.67 MB)
Non-trainable params: 896 (3.50 KB)
_________________________________________________________________
None
Training Model 1: MLP with BatchNorm...
Epoch 1/10
1563/1563 [==============================] - 13s 6ms/step - loss: 1.7347 - accuracy: 0.3832 - val_loss: 2.1473 - val_accuracy: 0.3034
Epoch 2/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.5232 - accuracy: 0.4553 - val_loss: 1.5415 - val_accuracy: 0.4547
Epoch 3/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.4411 - accuracy:
```

```
0.4864 - val_loss: 1.6450 - val_accuracy: 0.4087
Epoch 4/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.3848 - accuracy:
0.5071 - val_loss: 1.5184 - val_accuracy: 0.4530
Epoch 5/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.3353 - accuracy:
0.5241 - val_loss: 1.5725 - val_accuracy: 0.4592
Epoch 6/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.2948 - accuracy:
0.5386 - val_loss: 1.4949 - val_accuracy: 0.4582
Epoch 7/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.2556 - accuracy:
0.5538 - val_loss: 1.3717 - val_accuracy: 0.5053
Epoch 8/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.2233 - accuracy:
0.5654 - val_loss: 1.4717 - val_accuracy: 0.4853
Epoch 9/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.1933 - accuracy:
0.5770 - val_loss: 1.3859 - val_accuracy: 0.5016
Epoch 10/10
1563/1563 [==============================] - 8s 5ms/step - loss: 1.1593 - accuracy:
0.5863 - val_loss: 1.3731 - val_accuracy: 0.5116
313/313 [==============================] - 1s 3ms/step - loss: 1.3731 - accuracy: 0.5
116
Test accuracy for MLP with BatchNorm: 0.5116000175476074
Training Model 2: MLP with L2...
Epoch 1/10
1563/1563 [==============================] - 6s 4ms/step - loss: 2.1863 - accuracy:
0.2863 - val_loss: 1.8867 - val_accuracy: 0.3415
Epoch 2/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.9206 - accuracy:
0.3199 - val_loss: 1.8613 - val_accuracy: 0.3676
Epoch 3/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.9101 - accuracy:
0.3265 - val_loss: 1.8593 - val_accuracy: 0.3464
Epoch 4/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8926 - accuracy:
0.3337 - val_loss: 1.8471 - val_accuracy: 0.3685
Epoch 5/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8924 - accuracy:
0.3349 - val_loss: 1.7962 - val_accuracy: 0.3881
Epoch 6/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8865 - accuracy:
0.3353 - val_loss: 1.8175 - val_accuracy: 0.3823
Epoch 7/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8876 - accuracy:
0.3375 - val_loss: 1.7869 - val_accuracy: 0.3835
Epoch 8/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8756 - accuracy:
0.3406 - val_loss: 1.8704 - val_accuracy: 0.3443
Epoch 9/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.8839 - accuracy:
0.3344 - val_loss: 1.7968 - val_accuracy: 0.3650
Epoch 10/10
1563/1563 [==============================] - 5s 3ms/step - loss: 1.8740 - accuracy:
0.3367 - val_loss: 1.7492 - val_accuracy: 0.3970
313/313 [==============================] - 1s 2ms/step - loss: 1.7492 - accuracy: 0.3
970
Test accuracy for MLP with L2: 0.3970000147819519
Training Model 3: Simple CNN...
```

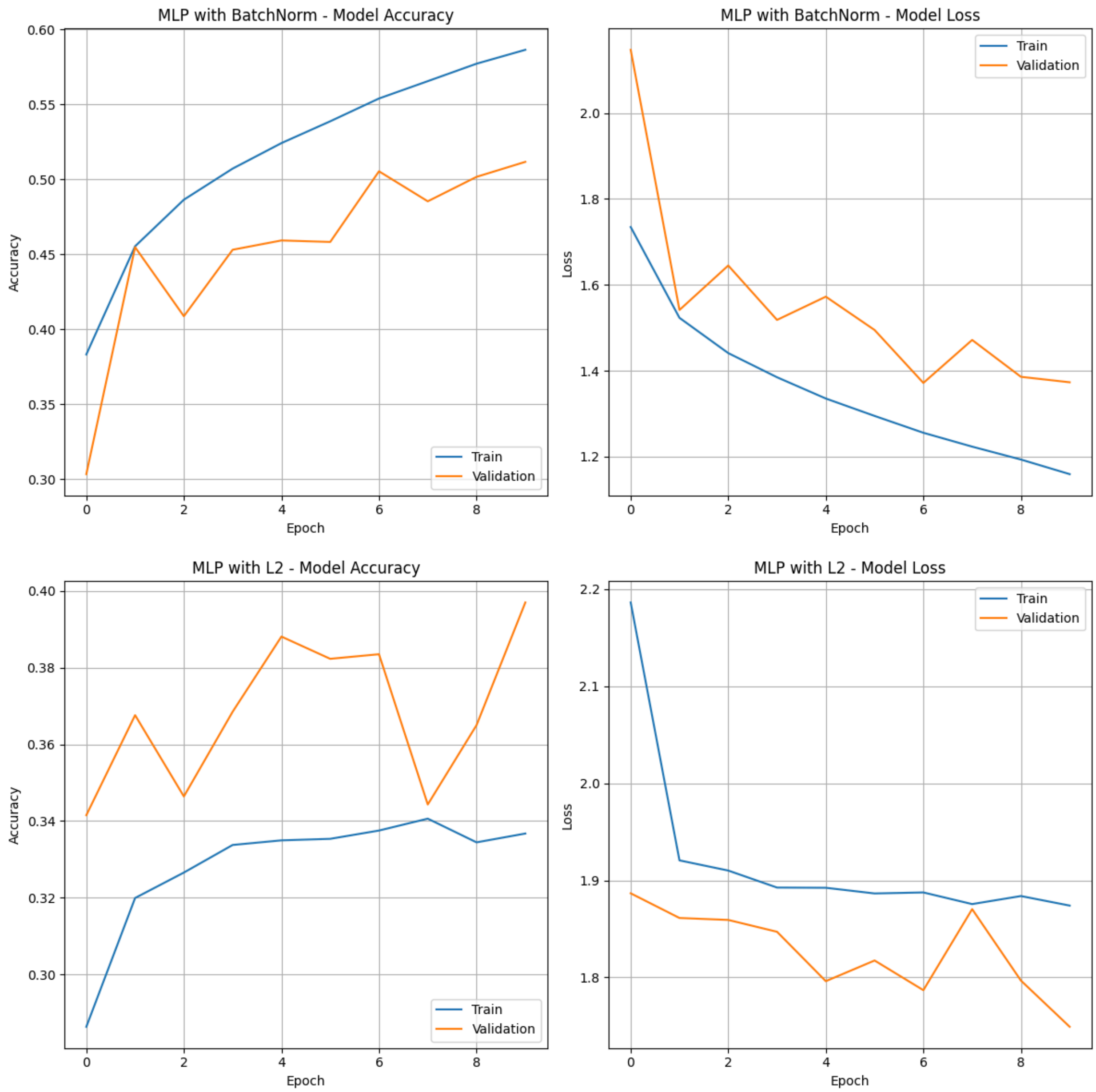
```
Epoch 1/10
1563/1563 [==============================] - 11s 4ms/step - loss: 1.4572 - accuracy:
0.4779 - val_loss: 1.2895 - val_accuracy: 0.5335
Epoch 2/10
1563/1563 [==============================] - 7s 5ms/step - loss: 1.1258 - accuracy:
0.6071 - val_loss: 1.0722 - val_accuracy: 0.6217
Epoch 3/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9933 - accuracy:
0.6532 - val_loss: 1.0358 - val_accuracy: 0.6477
Epoch 4/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.9107 - accuracy:
0.6848 - val_loss: 0.9552 - val_accuracy: 0.6718
Epoch 5/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.8413 - accuracy:
0.7080 - val_loss: 0.9666 - val_accuracy: 0.6644
Epoch 6/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.7812 - accuracy:
0.7292 - val_loss: 0.9419 - val_accuracy: 0.6815
Epoch 7/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.7333 - accuracy:
0.7445 - val_loss: 0.9341 - val_accuracy: 0.6884
Epoch 8/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.6869 - accuracy:
0.7604 - val_loss: 0.9366 - val_accuracy: 0.6888
Epoch 9/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.6418 - accuracy:
0.7754 - val_loss: 0.9704 - val_accuracy: 0.6838
Epoch 10/10
1563/1563 [==============================] - 7s 5ms/step - loss: 0.6015 - accuracy:
0.7897 - val_loss: 0.9458 - val_accuracy: 0.6909
313/313 [==============================] - 1s 2ms/step - loss: 0.9458 - accuracy: 0.6
909
Test accuracy for Simple CNN: 0.6909000277519226
Training Model 4: CNN with DA and Dropout...
Epoch 1/10
1563/1563 [==============================] - 9s 5ms/step - loss: 1.6409 - accuracy:
0.3993 - val_loss: 1.2958 - val_accuracy: 0.5445
Epoch 2/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.3169 - accuracy:
0.5320 - val_loss: 1.1419 - val_accuracy: 0.5916
Epoch 3/10
1563/1563 [==============================] - 7s 5ms/step - loss: 1.1903 - accuracy:
0.5821 - val_loss: 1.0418 - val_accuracy: 0.6381
Epoch 4/10
1563/1563 [==============================] - 7s 4ms/step - loss: 1.1090 - accuracy:
0.6100 - val_loss: 0.9715 - val_accuracy: 0.6582
Epoch 5/10
1563/1563 [==============================] - 7s 5ms/step - loss: 1.0510 - accuracy:
0.6313 - val_loss: 0.9415 - val_accuracy: 0.6731
Epoch 6/10
1563/1563 [==============================] - 7s 5ms/step - loss: 1.0090 - accuracy:
0.6480 - val_loss: 0.9243 - val_accuracy: 0.6779
Epoch 7/10
1563/1563 [==============================] - 7s 4ms/step - loss: 0.9588 - accuracy:
0.6638 - val_loss: 0.9108 - val_accuracy: 0.6806
Epoch 8/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.9260 - accuracy:
0.6760 - val_loss: 0.8999 - val_accuracy: 0.6857
Epoch 9/10
1563/1563 [==============================] - 7s 4ms/step - loss: 0.8957 - accuracy:
```

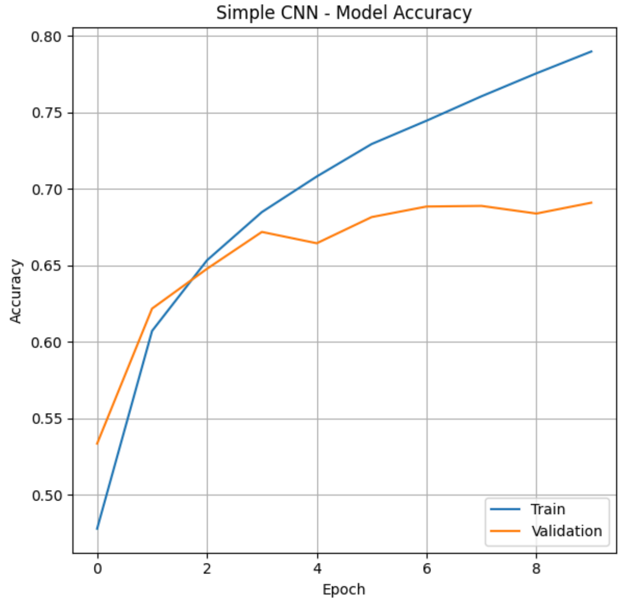
```
0.6848 - val_loss: 0.8889 - val_accuracy: 0.6946
Epoch 10/10
1563/1563 [==============================] - 8s 5ms/step - loss: 0.8745 - accuracy:
0.6923 - val_loss: 0.8904 - val_accuracy: 0.6956
313/313 [==============================] - 1s 2ms/step - loss: 0.8904 - accuracy: 0.6
956
Test accuracy for CNN with DA and Dropout: 0.6955999732017517
Training Model 5: ResNet...
Epoch 1/10
1563/1563 [==============================] - 18s 9ms/step - loss: 1.5953 - accuracy:
0.4298 - val_loss: 1.1129 - val_accuracy: 0.6007
Epoch 2/10
1563/1563 [==============================] - 12s 8ms/step - loss: 1.1086 - accuracy:
0.6112 - val_loss: 1.0145 - val_accuracy: 0.6411
Epoch 3/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.9156 - accuracy:
0.6877 - val_loss: 0.9534 - val_accuracy: 0.6802
Epoch 4/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.7776 - accuracy:
0.7380 - val_loss: 0.8449 - val_accuracy: 0.7255
Epoch 5/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.6726 - accuracy:
0.7745 - val_loss: 0.7478 - val_accuracy: 0.7575
Epoch 6/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.5857 - accuracy:
0.8028 - val_loss: 0.7808 - val_accuracy: 0.7399
Epoch 7/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.5164 - accuracy:
0.8273 - val_loss: 0.6081 - val_accuracy: 0.8064
Epoch 8/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.4532 - accuracy:
0.8476 - val_loss: 0.6316 - val_accuracy: 0.7951
Epoch 9/10
1563/1563 [==============================] - 12s 8ms/step - loss: 0.4013 - accuracy:
0.8648 - val_loss: 0.5905 - val_accuracy: 0.8172
Epoch 10/10
1563/1563 [==============================] - 12s 7ms/step - loss: 0.3594 - accuracy:
0.8782 - val_loss: 0.6309 - val_accuracy: 0.8137
313/313 [==============================] - 1s 3ms/step - loss: 0.6309 - accuracy: 0.8
137
Test accuracy for ResNet: 0.8137000203132629
```

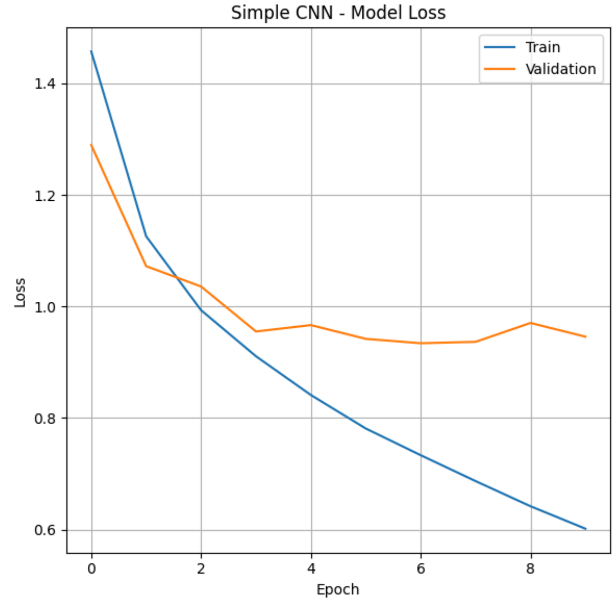
In [21]:

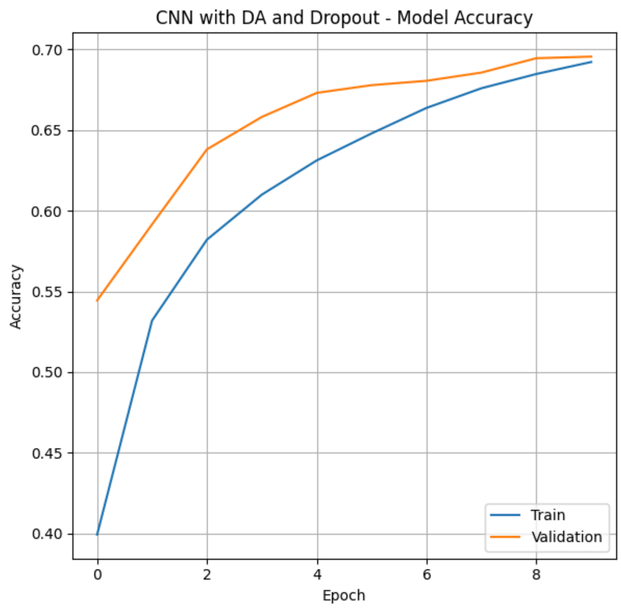
```python
import matplotlib.pyplot as plt

# Function to plot learning curves
def plot_learning_curves(history, title):
    plt.figure(figsize=(12, 6))

    # Plot training & validation accuracy values
    plt.subplot(1, 2, 1)
    plt.plot(history['accuracy'])
    plt.plot(history['val_accuracy'])
    plt.title(title + ' - Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'], loc='lower right')
    plt.grid(True)

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
```

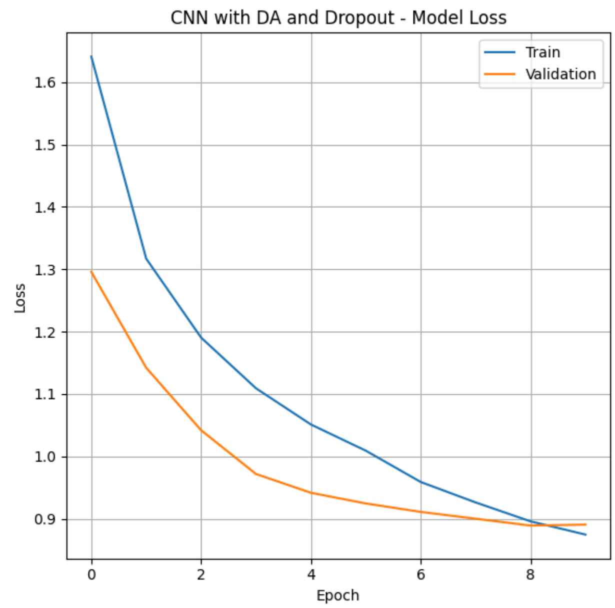
```python
    plt.plot(history['loss'])
    plt.plot(history['val_loss'])
    plt.title(title + ' - Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'], loc='upper right')
    plt.grid(True)

    plt.tight_layout()
    plt.show()

# Plot learning curves for each model
for history, name in zip(results, model_names):
    plot_learning_curves(history[0], name)
```

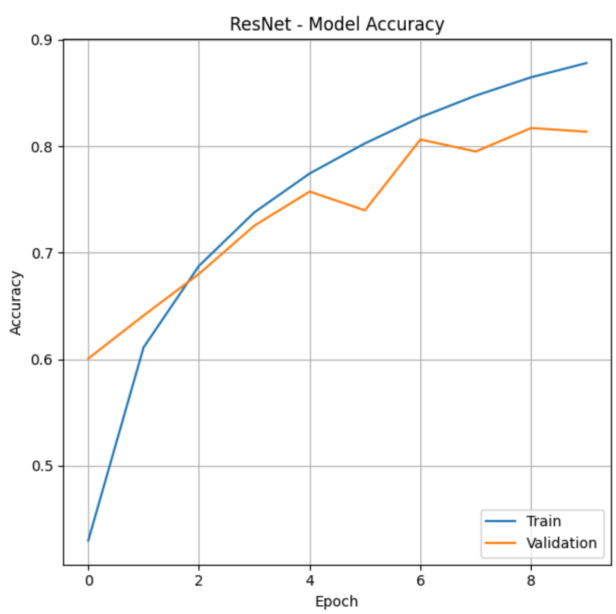


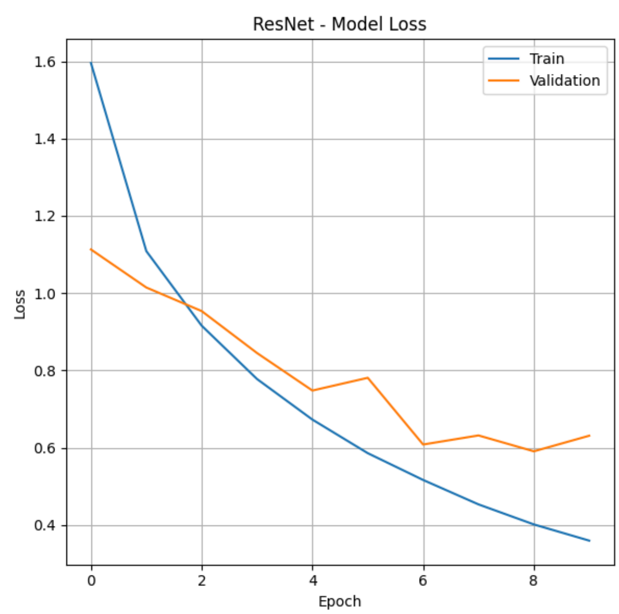

Simple CNN - Model Accuracy
Accuracy
Epoch
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0
2
4
6
8
Train
Validation


Simple CNN - Model Loss
Loss
Epoch
1.4
1.2
1.0
0.8
0.6
0
2
4
6
8
Train
Validation


CNN with DA and Dropout - Model Accuracy
Accuracy
Epoch
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0
2
4
6
8
Train
Validation


CNN with DA and Dropout - Model Loss
Loss
Epoch
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0
2
4
6
8
Train
Validation


ResNet - Model Accuracy
Accuracy
Epoch
0.9
0.8
0.7
0.6
0.5
0
2
4
6
8
Train
Validation


ResNet - Model Loss
Loss
Epoch
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
2
4
6
8
Train
Validation

Model Comparison

In [ ]:
```
# Calculate metrics for each model
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []

for name, model in zip(model_names, models):
    pred = np.argmax(model.predict(test_images), axis=1)
    accuracy = accuracy_score(test_labels, pred)
    precision = precision_score(test_labels, pred, average='macro')
    recall = recall_score(test_labels, pred, average='macro')
    f1 = f1_score(test_labels, pred, average='macro')

    accuracy_scores.append(accuracy)
    precision_scores.append(precision)
    recall_scores.append(recall)
    f1_scores.append(f1)

# Create a DataFrame to store the metrics
metrics_df = pd.DataFrame({
    'Model': model_names,
    'Accuracy': accuracy_scores,
    'Precision': precision_scores,
    'Recall': recall_scores,
    'F1 Score': f1_scores
})

# Print the metrics DataFrame
print("Metrics for each model:")
display(metrics_df)
```

```
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step
Metrics for each model:
```

	Model	Accuracy	Precision	Recall	F1 Score
0	MLP with BatchNorm	0.5116	0.536297	0.5116	0.511579
1	MLP with L2	0.3970	0.390330	0.3970	0.379442
2	Simple CNN	0.6909	0.692682	0.6909	0.687796
3	CNN with DA and Dropout	0.6956	0.695649	0.6956	0.692129
4	ResNet	0.8137	0.819477	0.8137	0.814185

In [ ]:
```
import matplotlib.pyplot as plt

# Plotting the metrics
metrics_df.set_index('Model', inplace=True)

plt.figure(figsize=(12, 8))
```

```
metrics_df.plot(kind='bar', colormap='viridis', alpha=0.8)
plt.title('Performance Metrics Comparison')
plt.xlabel('Model')
plt.ylabel('Score')
plt.xticks(rotation=45, ha='right')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.25), ncol=len(metrics_df.columr

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

```
<Figure size 1200x800 with 0 Axes>
```