AML FINAL PROJECT

Performance Evaluation of Various Deep Learning Architectures on the CIFAR-10 Dataset

```
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

# Load and preprocess the CIFAR-10 dataset
(train_imgs, train_lbls), (test_imgs, test_lbls) = datasets.cifar10.load_data()
train_imgs = train_imgs / 255.0
test_imgs = test_imgs / 255.0

# Define models
model_mlp_bn = 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')
])

model_mlp_l2 = 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')
])

model_cnn_simple = 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')
])

model_cnn_aug_dropout = 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')
])

model_res_net = 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.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')
])

# Compile all models
model_collection = [model_mlp_bn, model_mlp_l2, model_cnn_simple, model_cnn_aug_dropout, model_res_net]
model_labels = ["MLP with BatchNorm", "MLP with L2", "Simple CNN", "CNN with DA and Dropout", "ResNet"]

# Display model summaries
for i, model in enumerate(model_collection):
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    print(model_labels[i] + " Model Architecture:\n")
    model.summary()

# Train all models and store results
training_outcomes = []

for idx, (model, label) in enumerate(zip(model_collection, model_labels), start=1):
    print(f"Training Model {idx}: {label}...")
    history = model.fit(train_imgs, train_lbls, epochs=10, validation_data=(test_imgs, test_lbls))

    # Evaluate the model
    test_loss, test_acc = model.evaluate(test_imgs, test_lbls)
    print(f'Test accuracy for {label}:', test_acc)
    training_outcomes.append((history.history, test_loss, test_acc))
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
MLP with BatchNorm Model Architecture :
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 512)	1,573,376
batch_normalization (BatchNormalization)	(None, 512)	2,048
activation (Activation)	(None, 512)	0
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
batch_normalization_1 (BatchNormalization)	(None, 256)	1,024
activation_1 (Activation)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2,570

Total params: 1,710,346 (6.52 MB)
Trainable params: 1,708,810 (6.52 MB)
Non-trainable params: 1,536 (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)	1,573,376
dropout_2 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 10)	5,130

Total params: 1,578,506 (6.02 MB)
Trainable params: 1,578,506 (6.02 MB)
Non-trainable params: 0 (0.00 B)
None
Simple CNN Model Architecture :

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten_2 (Flatten)	(None, 2304)	0
dense_5 (Dense)	(None, 64)	147,520
dense_6 (Dense)	(None, 10)	650

Total params: 167,562 (654.54 KB)
Trainable params: 167,562 (654.54 KB)
Non-trainable params: 0 (0.00 B)
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 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_3 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten_3 (Flatten)	(None, 2304)	0
dense_7 (Dense)	(None, 128)	295,040
dropout_3 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 10)	1,290

Total params: 315,722 (1.20 MB)
Trainable params: 315,722 (1.20 MB)
Non-trainable params: 0 (0.00 B)
None
ResNet Model Architecture :

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 30, 30, 32)	896
batch_normalization_2 (BatchNormalization)	(None, 30, 30, 32)	128
conv2d_5 (Conv2D)	(None, 30, 30, 32)	9,248
batch_normalization_3 (BatchNormalization)	(None, 30, 30, 32)	128
max_pooling2d_4 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_6 (Conv2D)	(None, 15, 15, 64)	18,496
batch_normalization_4 (BatchNormalization)	(None, 15, 15, 64)	256
conv2d_7 (Conv2D)	(None, 15, 15, 64)	36,928
batch_normalization_5 (BatchNormalization)	(None, 15, 15, 64)	256
max_pooling2d_5 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_8 (Conv2D)	(None, 7, 7, 128)	73,856
batch_normalization_6 (BatchNormalization)	(None, 7, 7, 128)	512
conv2d_9 (Conv2D)	(None, 7, 7, 128)	147,584
batch_normalization_7 (BatchNormalization)	(None, 7, 7, 128)	512
max_pooling2d_6 (MaxPooling2D)	(None, 3, 3, 128)	0
flatten_4 (Flatten)	(None, 1152)	0
dense_9 (Dense)	(None, 128)	147,584
dropout_4 (Dropout)	(None, 128)	0
dense_10 (Dense)	(None, 10)	1,290

Total params: 437,674 (1.67 MB)
Trainable params: 436,778 (1.67 MB)
Non-trainable params: 896 (3.50 KB)

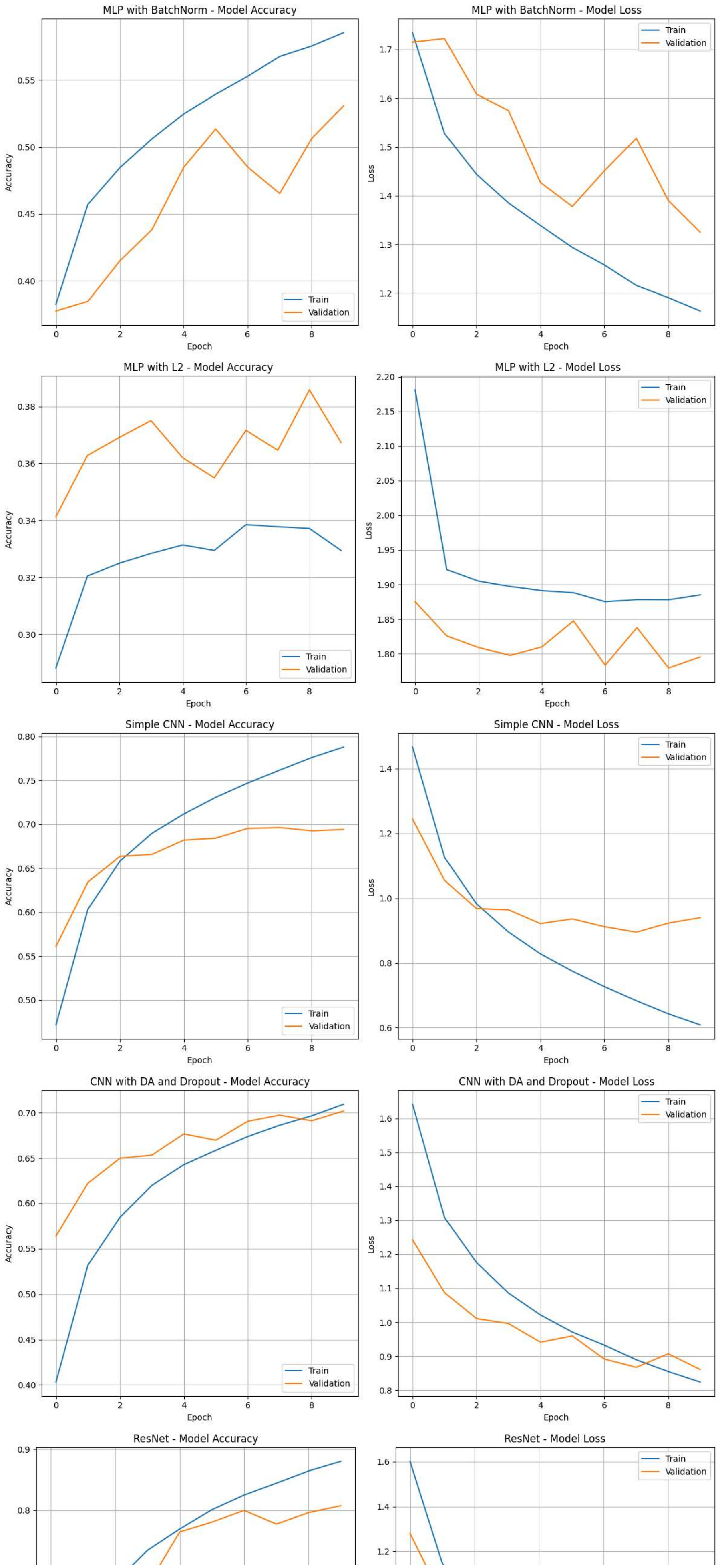
```
None
Training Model 1: MLP with BatchNorm...
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 35ms/step - accuracy: 0.3400 - loss: 1.8736 - val_accuracy: 0.3776 - val_loss: 1.7150
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 71s 28ms/step - accuracy: 0.4511 - loss: 1.5436 - val_accuracy: 0.3848 - val_loss: 1.7223
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 43s 28ms/step - accuracy: 0.4836 - loss: 1.4426 - val_accuracy: 0.4150 - val_loss: 1.6081
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 28ms/step - accuracy: 0.5033 - loss: 1.3873 - val_accuracy: 0.4382 - val_loss: 1.5748
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 28ms/step - accuracy: 0.5251 - loss: 1.3343 - val_accuracy: 0.4849 - val_loss: 1.4272
Epoch 6/10
```

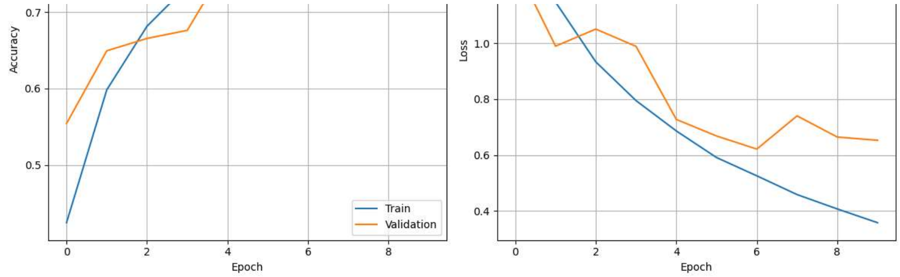
```
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 28ms/step - accuracy: 0.5405 - loss: 1.2928 - val_accuracy: 0.5135 - val_loss: 1.3778
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 28ms/step - accuracy: 0.5528 - loss: 1.2559 - val_accuracy: 0.4851 - val_loss: 1.4509
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 84s 29ms/step - accuracy: 0.5696 - loss: 1.2043 - val_accuracy: 0.4653 - val_loss: 1.5177
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 29ms/step - accuracy: 0.5791 - loss: 1.1780 - val_accuracy: 0.5063 - val_loss: 1.3904
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 29ms/step - accuracy: 0.5940 - loss: 1.1479 - val_accuracy: 0.5308 - val_loss: 1.3250
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.5365 - loss: 1.3206
Test accuracy for MLP with BatchNorm: 0.5307999849319458
Training Model 2: MLP with L2...
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 52s 32ms/step - accuracy: 0.2554 - loss: 2.5250 - val_accuracy: 0.3413 - val_loss: 1.8753
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 48s 31ms/step - accuracy: 0.3197 - loss: 1.9242 - val_accuracy: 0.3628 - val_loss: 1.8257
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 51s 33ms/step - accuracy: 0.3301 - loss: 1.9002 - val_accuracy: 0.3691 - val_loss: 1.8090
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 32ms/step - accuracy: 0.3276 - loss: 1.8962 - val_accuracy: 0.3750 - val_loss: 1.7976
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 31ms/step - accuracy: 0.3303 - loss: 1.8908 - val_accuracy: 0.3620 - val_loss: 1.8100
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 49s 31ms/step - accuracy: 0.3294 - loss: 1.8840 - val_accuracy: 0.3549 - val_loss: 1.8475
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 32ms/step - accuracy: 0.3384 - loss: 1.8745 - val_accuracy: 0.3716 - val_loss: 1.7834
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 31ms/step - accuracy: 0.3399 - loss: 1.8761 - val_accuracy: 0.3646 - val_loss: 1.8377
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 32ms/step - accuracy: 0.3330 - loss: 1.8821 - val_accuracy: 0.3858 - val_loss: 1.7794
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 49s 31ms/step - accuracy: 0.3320 - loss: 1.8834 - val_accuracy: 0.3673 - val_loss: 1.7955
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3698 - loss: 1.7966
Test accuracy for MLP with L2: 0.36730000376701355
Training Model 3: Simple CNN...
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 75s 47ms/step - accuracy: 0.3860 - loss: 1.6860 - val_accuracy: 0.5611 - val_loss: 1.2442
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 70s 45ms/step - accuracy: 0.5900 - loss: 1.1589 - val_accuracy: 0.6344 - val_loss: 1.0551
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 69s 44ms/step - accuracy: 0.6546 - loss: 0.9860 - val_accuracy: 0.6635 - val_loss: 0.9679
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 44ms/step - accuracy: 0.6908 - loss: 0.8986 - val_accuracy: 0.6657 - val_loss: 0.9642
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 84s 45ms/step - accuracy: 0.7120 - loss: 0.8253 - val_accuracy: 0.6820 - val_loss: 0.9216
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 44ms/step - accuracy: 0.7350 - loss: 0.7656 - val_accuracy: 0.6842 - val_loss: 0.9360
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 44ms/step - accuracy: 0.7501 - loss: 0.7216 - val_accuracy: 0.6953 - val_loss: 0.9122
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 71s 45ms/step - accuracy: 0.7653 - loss: 0.6709 - val_accuracy: 0.6963 - val_loss: 0.8952
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 44ms/step - accuracy: 0.7822 - loss: 0.6297 - val_accuracy: 0.6925 - val_loss: 0.9230
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 70s 45ms/step - accuracy: 0.7925 - loss: 0.5919 - val_accuracy: 0.6941 - val_loss: 0.9400
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.6981 - loss: 0.9343
Test accuracy for Simple CNN: 0.694100022315979
Training Model 4: CNN with DA and Dropout...
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 74s 47ms/step - accuracy: 0.3227 - loss: 1.8473 - val_accuracy: 0.5640 - val_loss: 1.2422
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 73s 47ms/step - accuracy: 0.5188 - loss: 1.3408 - val_accuracy: 0.6223 - val_loss: 1.0872
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 72s 46ms/step - accuracy: 0.5808 - loss: 1.1807 - val_accuracy: 0.6498 - val_loss: 1.0109
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 47ms/step - accuracy: 0.6196 - loss: 1.0879 - val_accuracy: 0.6532 - val_loss: 0.9962
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 47ms/step - accuracy: 0.6440 - loss: 1.0221 - val_accuracy: 0.6766 - val_loss: 0.9413
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 85s 49ms/step - accuracy: 0.6561 - loss: 0.9771 - val_accuracy: 0.6696 - val_loss: 0.9597
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 49ms/step - accuracy: 0.6721 - loss: 0.9311 - val_accuracy: 0.6905 - val_loss: 0.8917
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 87s 52ms/step - accuracy: 0.6883 - loss: 0.8860 - val_accuracy: 0.6973 - val_loss: 0.8673
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 75s 48ms/step - accuracy: 0.7016 - loss: 0.8439 - val_accuracy: 0.6911 - val_loss: 0.9071
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 48ms/step - accuracy: 0.7132 - loss: 0.8131 - val_accuracy: 0.7019 - val_loss: 0.8608
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - accuracy: 0.7036 - loss: 0.8482
Test accuracy for CNN with DA and Dropout: 0.7019000053405762
Training Model 5: ResNet...
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 404s 255ms/step - accuracy: 0.3422 - loss: 1.8599 - val_accuracy: 0.5541 - val_loss: 1.2803
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 441s 254ms/step - accuracy: 0.5742 - loss: 1.2072 - val_accuracy: 0.6495 - val_loss: 0.9894
```

```
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 399s 255ms/step - accuracy: 0.6711 - loss: 0.9615 - val_accuracy: 0.6657 - val_loss: 1.0505
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 395s 253ms/step - accuracy: 0.7294 - loss: 0.8073 - val_accuracy: 0.6763 - val_loss: 0.9887
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 443s 254ms/step - accuracy: 0.7617 - loss: 0.7101 - val_accuracy: 0.7645 - val_loss: 0.7276
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 446s 257ms/step - accuracy: 0.8006 - loss: 0.5866 - val_accuracy: 0.7805 - val_loss: 0.6684
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 399s 256ms/step - accuracy: 0.8277 - loss: 0.5167 - val_accuracy: 0.7999 - val_loss: 0.6216
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 455s 264ms/step - accuracy: 0.8491 - loss: 0.4414 - val_accuracy: 0.7773 - val_loss: 0.7403
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 430s 256ms/step - accuracy: 0.8692 - loss: 0.3903 - val_accuracy: 0.7963 - val_loss: 0.6646
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 452s 263ms/step - accuracy: 0.8890 - loss: 0.3322 - val_accuracy: 0.8075 - val_loss: 0.6531
313/313 ━━━━━━━━━━━━━━━━━━━━ 22s 69ms/step - accuracy: 0.8094 - loss: 0.6482
Test accuracy for ResNet: 0.8075000047683716
```

```python
# Function to plot learning curves
def display_learning_curves(history, model_name):
    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(model_name + ' - 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)
    plt.plot(history['loss'])
    plt.plot(history['val_loss'])
    plt.title(model_name + ' - 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, model_name in zip(training_outcomes, model_labels):
    display_learning_curves(history[0], model_name)

# Calculate metrics for each model
acc_scores = []
prec_scores = []
rec_scores = []
f1_scores = []

for model_name, model in zip(model_labels, model_collection):
    preds = np.argmax(model.predict(test_imgs), axis=1)
    accuracy = accuracy_score(test_lbls, preds)
    precision = precision_score(test_lbls, preds, average='macro')
    recall = recall_score(test_lbls, preds, average='macro')
    f1 = f1_score(test_lbls, preds, average='macro')

    acc_scores.append(accuracy)
    prec_scores.append(precision)
    rec_scores.append(recall)
    f1_scores.append(f1)

# Create a DataFrame to store the metrics
metrics_data = pd.DataFrame({
    'Model': model_labels,
    'Accuracy': acc_scores,
    'Precision': prec_scores,
    'Recall': rec_scores,
    'F1 Score': f1_scores
})

# Print the metrics DataFrame
```

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

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

plt.figure(figsize=(12, 8))
metrics_data.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_data.columns))

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


MLP with BatchNorm - Model Accuracy
Accuracy
Epoch
Train
Validation
MLP with BatchNorm - Model Loss
Loss
Epoch
Train
Validation
MLP with L2 - Model Accuracy
Accuracy
Epoch
Train
Validation
MLP with L2 - Model Loss
Loss
Epoch
Train
Validation
Simple CNN - Model Accuracy
Accuracy
Epoch
Train
Validation
Simple CNN - Model Loss
Loss
Epoch
Train
Validation
CNN with DA and Dropout - Model Accuracy
Accuracy
Epoch
Train
Validation
CNN with DA and Dropout - Model Loss
Loss
Epoch
Train
Validation
ResNet - Model Accuracy
ResNet - Model Loss
Train
Validation




```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 22s 69ms/step
Metrics for each model:
                     Model  Accuracy  Precision  Recall  F1 Score
0       MLP with BatchNorm    0.5308   0.532260  0.5308  0.519068
1              MLP with L2    0.3673   0.407799  0.3673  0.351644
2               Simple CNN    0.6941   0.699398  0.6941  0.694818
3  CNN with DA and Dropout    0.7019   0.701438  0.7019  0.699712
4                   ResNet    0.8075   0.811879  0.8075  0.807482
<Figure size 1200x800 with 0 Axes>
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.