Exploring Convolutional Neural Networks: Comparing AlexNet vs VGG16

Daniel Coblentz

CHAPTER ONE

INTRODUCTION

BACKGROUND OF THE STUDY

INTRODUCTION

This project focuses on comparing and analyzing two widely-known convolutional neural network (CNN) models, VGG16 and AlexNet, to evaluate their performance on various image classification tasks. The Goal is to explore how different architectures and training techniques impact model accuracy and performance.

BACKGROUND OF THE STUDY

Convolutional neural networks have become the backbone of many computer vision tasks, such as image classification, object detection, and facial recognition. Two foundational models in this domain are VGG16 and AlexNet. Introduced in 2012, AlexNet made a breakthrough by achieving high performance in the ImageNet challenge, demonstrating the power of deep learning on large-scale image recognition. It introduced techniques like ReLU activation and dropout regularization. The other architecture, VGG16, built on this by using a deeper, more uniform structure of 3x3 convolution kernels, achieving high accuracy but with the trade-off of needing more computational resources.

STATEMENT OF THE PROBLEM

1. **Evaluating Model Effectiveness:** How do the performance, accuracy, and training time of VGG16 and AlexNet differ when trained on the same datasets?

2. **Impact of Data Augmentation:** What is the effect of applying data augmentation techniques (rotation, flipping, scaling) on model performance and generalization?

3. **Transfer Learning Efficiency:** Does using pre-trained weights (transfer learning) improve the performance of VGG16 compared to training AlexNet from scratch?

4. **Model Complexity vs. Accuracy Trade-off:** What are the trade-offs between model complexity, computational cost, and classification accuracy for VGG16 and AlexNet?

5. **Hyperparameter Optimization:** How do variations in learning rates, batch sizes, and optimizer choices affect model accuracy and convergence?

Project Scope

Scope: This project focuses on classifying static hand gestures from a predefined dataset using a deep learning model. The study is limited to the gestures present in the dataset and evaluates performance based on model accuracy and loss metrics. The implementation uses AlexNet and VGG16 architecture along with various modifications to the base models provided.

CHAPTER TWO

Review of Related Studies and Literature

Related Literature

Adithya, V., & Rajesh, R. (2020). A Deep Convolutional Neural Network Approach for Static Hand Gesture Recognition. Link to paper

Pavlo, M., Shalini, G., Kihwan, K., & Jan, K. (n.d.). Hand Gesture Recognition with 3D Convolutional Neural Networks. Link to paper

Zahirul, I., Mohammad, S., Raihan, U., Karl, A. Static Hand Gesture Recognition Using Convolutional neural Network With Data Augmentation. Link to paper

Related Studies

Dive into Deep Learning: Link to paper

Built-In: Link to paper

Medium: Link to paper

CHAPTER THREE

Methodology

Data preprocessing

Techniques: Image resizing:
AlexNet: (227x227 x 3)
VGG16(224x224 x 3)
normalization

Model Design

Implemented Layer configurations and to evaluate performance on selected datasets.

Data Augmentation

Applied techniques such as rotation, zoom and horizontal flipping to reduce overfitting.

Hyperparameter Tuning

Tested different learning rates, and optimizers to determine their effects on training speed and overall performance.

Training

Each model was trained on an average of 10 epochs, using early stopping based on validation loss.

Data Gathering Procedure

- The dataset was collected from Kaggle and contains images organized into multiple classes for classification tasks. Each image was preprocessed by resizing it to the appropriate dimensions for the models and normalizing pixel values. The dataset was then split into a training set using a 70/30 split. Additionally, data augmentation techniques were applied to the training set to increase variability and robustness during training.

- Link to dataset: Hand_Gesture_Recognition

Data augmentation

AlexNet:
- Rotation: Randomly rotated images by up to 20 degrees.
- Scaling: Applied scaling of up to 20 percent of original size.
- Horizontal Flip: Randomly flipped images horizontally.

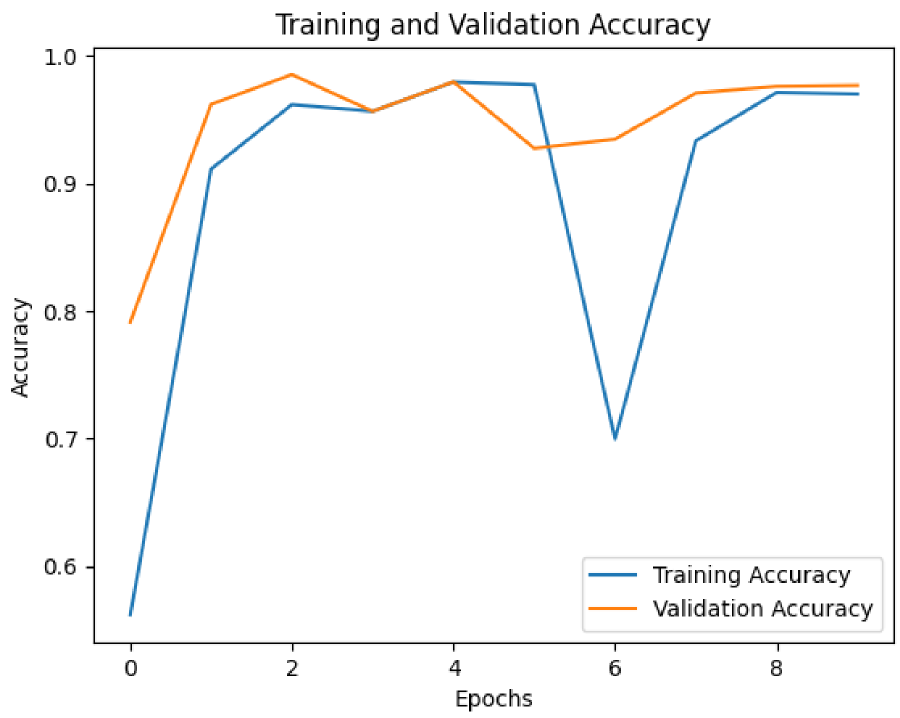
VGG16:
- Rotation: Randomly rotated images by up to 20 degrees.
- Scaling: Applied scaling of up to 20 percent of original size.
- Horizontal Flip: Randomly flipped images horizontally.

CHAPTER FOUR

Presentation, Interpretation, and Analysis Of Data

Presentation Of Data

AlexNet Base Model stats:




```
 Total params: 58,322,314 (222.48 MB)
 Trainable params: 58,322,314 (222.48 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 78s 154ms/step - accuracy: 0.3976 - loss: 12.5398 - val_accuracy: 0.7912 - val_loss: 0.5459
Epoch 2/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 56s 114ms/step - accuracy: 0.8800 - loss: 0.3575 - val_accuracy: 0.9620 - val_loss: 0.1202
Epoch 3/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 83s 117ms/step - accuracy: 0.9553 - loss: 0.1362 - val_accuracy: 0.9853 - val_loss: 0.0327
Epoch 4/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 52s 120ms/step - accuracy: 0.9609 - loss: 0.1355 - val_accuracy: 0.9565 - val_loss: 0.1203
Epoch 5/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 80s 114ms/step - accuracy: 0.9758 - loss: 0.0753 - val_accuracy: 0.9797 - val_loss: 0.0690
Epoch 6/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 82s 114ms/step - accuracy: 0.9805 - loss: 0.0602 - val_accuracy: 0.9275 - val_loss: 0.5038
Epoch 7/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 82s 114ms/step - accuracy: 0.5829 - loss: 1.4730 - val_accuracy: 0.9347 - val_loss: 0.2263
Epoch 8/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 61s 138ms/step - accuracy: 0.9145 - loss: 0.2546 - val_accuracy: 0.9708 - val_loss: 0.1427
Epoch 9/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 80s 134ms/step - accuracy: 0.9724 - loss: 0.0837 - val_accuracy: 0.9762 - val_loss: 0.0594
Epoch 10/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 58s 133ms/step - accuracy: 0.9731 - loss: 0.0746 - val_accuracy: 0.9768 - val_loss: 0.0794
188/188 - 13s - 67ms/step - accuracy: 0.9768 - loss: 0.0794
Validation accuracy: 0.9768333435058594
```

Different Learning Rates & early stopping:

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
438/438 ━━━━━━━━━━━━━━━━━━━━ 70s 148ms/step - accuracy: 0.3328 - loss: 3.5083 - val_accuracy: 0.9515 - val_loss: 0.2546
Epoch 2/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 61s 140ms/step - accuracy: 0.8679 - loss: 0.4151 - val_accuracy: 0.9878 - val_loss: 0.0454
Epoch 3/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 64s 147ms/step - accuracy: 0.9597 - loss: 0.1443 - val_accuracy: 0.9928 - val_loss: 0.0200
Epoch 4/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 72s 164ms/step - accuracy: 0.9767 - loss: 0.0803 - val_accuracy: 0.9925 - val_loss: 0.0207
Epoch 5/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 61s 139ms/step - accuracy: 0.9800 - loss: 0.0667 - val_accuracy: 0.9887 - val_loss: 0.0299
Epoch 6/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 70s 159ms/step - accuracy: 0.9817 - loss: 0.0553 - val_accuracy: 0.9918 - val_loss: 0.0180
Epoch 7/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 65s 148ms/step - accuracy: 0.9877 - loss: 0.0396 - val_accuracy: 0.9900 - val_loss: 0.0251
Epoch 8/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 64s 146ms/step - accuracy: 0.9881 - loss: 0.0396 - val_accuracy: 0.9945 - val_loss: 0.0115
Epoch 9/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 83s 149ms/step - accuracy: 0.9873 - loss: 0.0352 - val_accuracy: 0.9945 - val_loss: 0.0114
Epoch 10/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 67s 153ms/step - accuracy: 0.9889 - loss: 0.0330 - val_accuracy: 0.9948 - val_loss: 0.0144
188/188 - 14s - 72ms/step - accuracy: 0.9948 - loss: 0.0144
Validation accuracy for Learning Rate = 1e-05: 0.9948

Validation accuracy for different learning rates:
Learning Rate = 0.01: Validation Accuracy = 0.0993
Learning Rate = 0.001: Validation Accuracy = 0.0965
Learning Rate = 0.0001: Validation Accuracy = 0.9900
Learning Rate = 1e-05: Validation Accuracy = 0.9948
```

```python
# Define a list of learning rates to experiment with
learning_rates = [0.01, 0.001, 0.0001, 0.00001]

# Dictionary to store validation accuracies for each learning rate
results = {}

# Implement Early Stopping to avoid overfitting
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Loop through each learning rate and train the model
for lr in learning_rates:
    print(f"\nTraining with Learning Rate = {lr}\n")

    # Build a new AlexNet model for each learning rate
    alexnet_model = build_alexnet()

    # Compile the model with the specified learning rate
    alexnet_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr), loss='categorical_crossentropy', metrics=['accuracy'])
```

Data augmentation results:

- Flip

Validation accuracy: 65%
Average training time: 71.1 sec
Total training time: 11.85 min

```
438/438 ━━━━━━━━━━━━━━━━━━━━ 75s 162ms/step - accuracy: 0.3697 - loss: 1.6703 - val_accuracy: 0.4423 - val_loss: 2.5347
Epoch 2/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 67s 152ms/step - accuracy: 0.9727 - loss: 0.0875 - val_accuracy: 0.5745 - val_loss: 2.1642
Epoch 3/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 69s 154ms/step - accuracy: 0.9780 - loss: 0.0525 - val_accuracy: 0.6028 - val_loss: 2.2225
Epoch 4/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 66s 150ms/step - accuracy: 0.9803 - loss: 0.0439 - val_accuracy: 0.6372 - val_loss: 2.2195
Epoch 5/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 83s 153ms/step - accuracy: 0.9877 - loss: 0.0253 - val_accuracy: 0.6245 - val_loss: 2.3823
Epoch 6/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 65s 147ms/step - accuracy: 0.9882 - loss: 0.0223 - val_accuracy: 0.5303 - val_loss: 2.1457
Epoch 7/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 67s 152ms/step - accuracy: 0.9839 - loss: 0.0324 - val_accuracy: 0.7245 - val_loss: 1.4721
Epoch 8/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 82s 151ms/step - accuracy: 0.9890 - loss: 0.0198 - val_accuracy: 0.7132 - val_loss: 1.8181
Epoch 9/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 67s 151ms/step - accuracy: 0.9883 - loss: 0.0198 - val_accuracy: 0.6798 - val_loss: 2.2021
Epoch 10/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 80s 147ms/step - accuracy: 0.9868 - loss: 0.0217 - val_accuracy: 0.6518 - val_loss: 2.2797
188/188 - 18s - 96ms/step - accuracy: 0.6510 - loss: 2.2491
Validation accuracy after flipping augmentation: 0.6510
```

- Rotation

Validation accuracy: 78%
Average training time: 321.4 sec
Total training time: 26.78 min

```
Epoch 1/10
438/438 [==============================] - 322s 730ms/step - loss: 1.2871 - accuracy: 0.6571 - val_loss: 18.5222 - val_accuracy: 0.1103
Epoch 2/10
438/438 [==============================] - 323s 737ms/step - loss: 0.1876 - accuracy: 0.9359 - val_loss: 1.0166 - val_accuracy: 0.7877
Epoch 3/10
438/438 [==============================] - 320s 730ms/step - loss: 0.1247 - accuracy: 0.9619 - val_loss: 3.0698 - val_accuracy: 0.5620
Epoch 4/10
438/438 [==============================] - 321s 733ms/step - loss: 0.1081 - accuracy: 0.9672 - val_loss: 2.2272 - val_accuracy: 0.6977
Epoch 5/10
438/438 [==============================] - 321s 732ms/step - loss: 0.1081 - accuracy: 0.9684 - val_loss: 2.0637 - val_accuracy: 0.6345
188/188 - 72s - loss: 1.0659 - accuracy: 0.7822 - 72s/epoch - 384ms/step
Validation accuracy after rotation: 0.7822
```

- Scaling

Validation accuracy: 77%

Average training time: 308.9 sec

Total training time: 51 min

```
Epoch 1/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 295s 657ms/step - accuracy: 0.4904 - loss: 1.3912 - val_accuracy: 0.5252 - val_loss: 2.6087
Epoch 2/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 315s 645ms/step - accuracy: 0.9733 - loss: 0.0765 - val_accuracy: 0.6435 - val_loss: 1.7488
Epoch 3/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 295s 665ms/step - accuracy: 0.9856 - loss: 0.0354 - val_accuracy: 0.5947 - val_loss: 1.9471
Epoch 4/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 303s 684ms/step - accuracy: 0.9806 - loss: 0.0422 - val_accuracy: 0.6548 - val_loss: 2.1511
Epoch 5/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 312s 704ms/step - accuracy: 0.9850 - loss: 0.0286 - val_accuracy: 0.6110 - val_loss: 2.4474
Epoch 6/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 311s 700ms/step - accuracy: 0.9844 - loss: 0.0346 - val_accuracy: 0.6393 - val_loss: 1.9196
Epoch 7/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 365s 826ms/step - accuracy: 0.9840 - loss: 0.0251 - val_accuracy: 0.7113 - val_loss: 1.6487
Epoch 8/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 295s 665ms/step - accuracy: 0.9870 - loss: 0.0222 - val_accuracy: 0.6815 - val_loss: 2.2315
Epoch 9/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 297s 671ms/step - accuracy: 0.9882 - loss: 0.0221 - val_accuracy: 0.7737 - val_loss: 1.1634
Epoch 10/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 301s 678ms/step - accuracy: 0.9863 - loss: 0.0219 - val_accuracy: 0.7775 - val_loss: 1.4410
188/188 - 87s - 464ms/step - accuracy: 0.7775 - loss: 1.4302
Validation accuracy after scaling augmentation: 0.7775
```

- [Auto augmentation]()

Batch size results:

* With batchnormization() applied

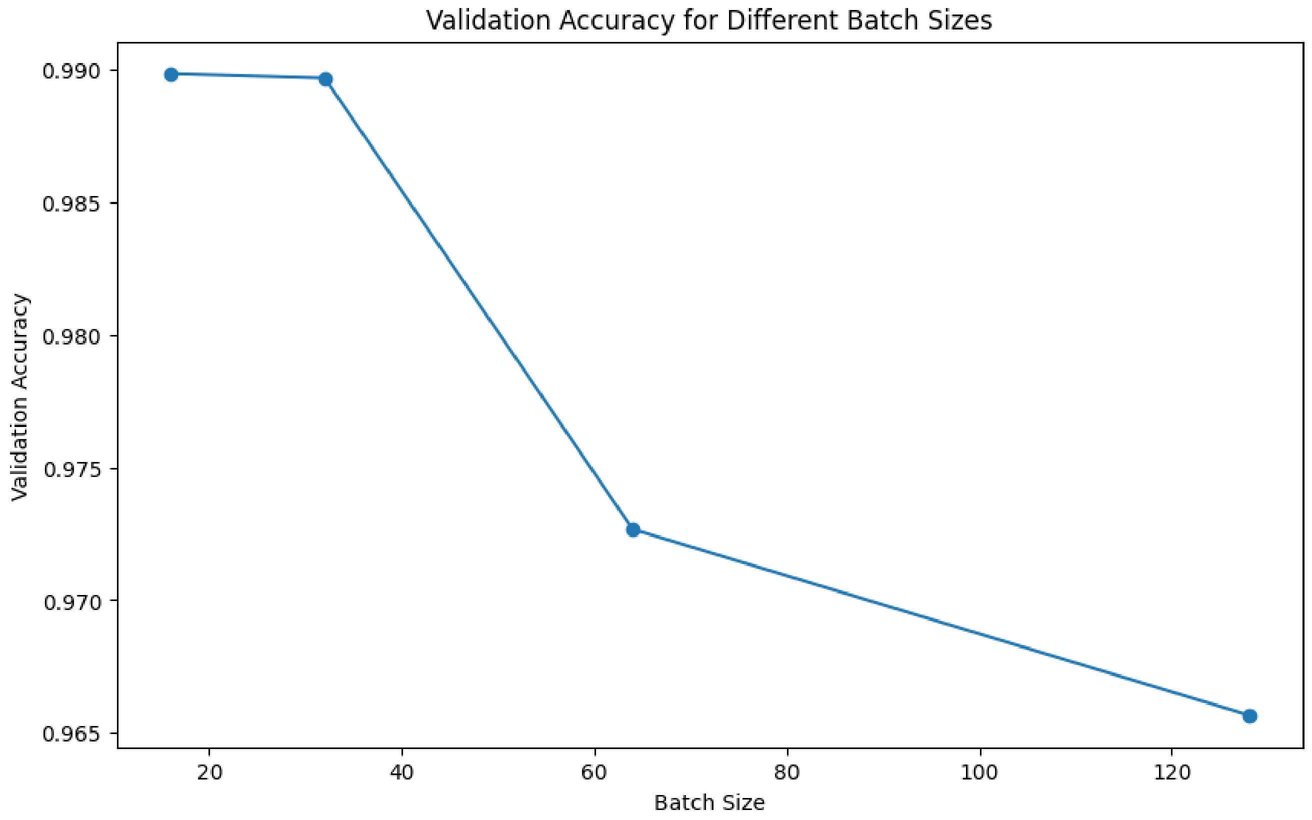
Tested sizes:

16 = 0.9898

32 = 0.9897

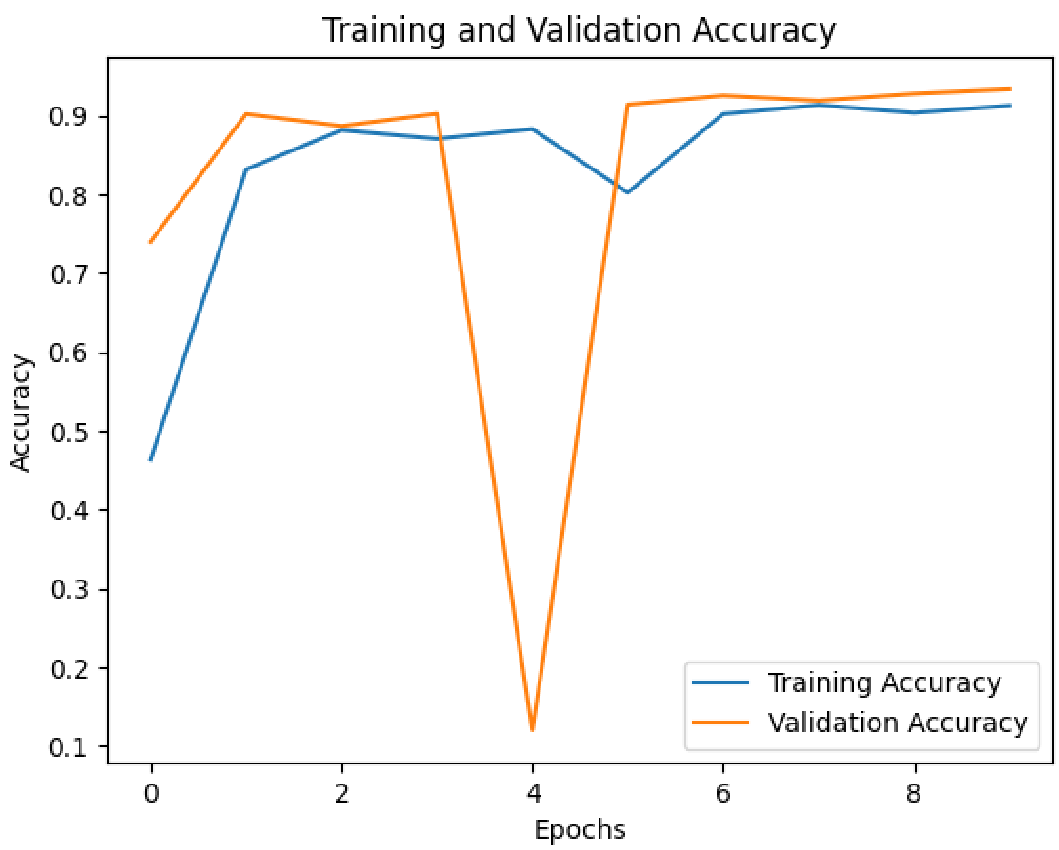
64 = 0.9727

128 = 0.9657

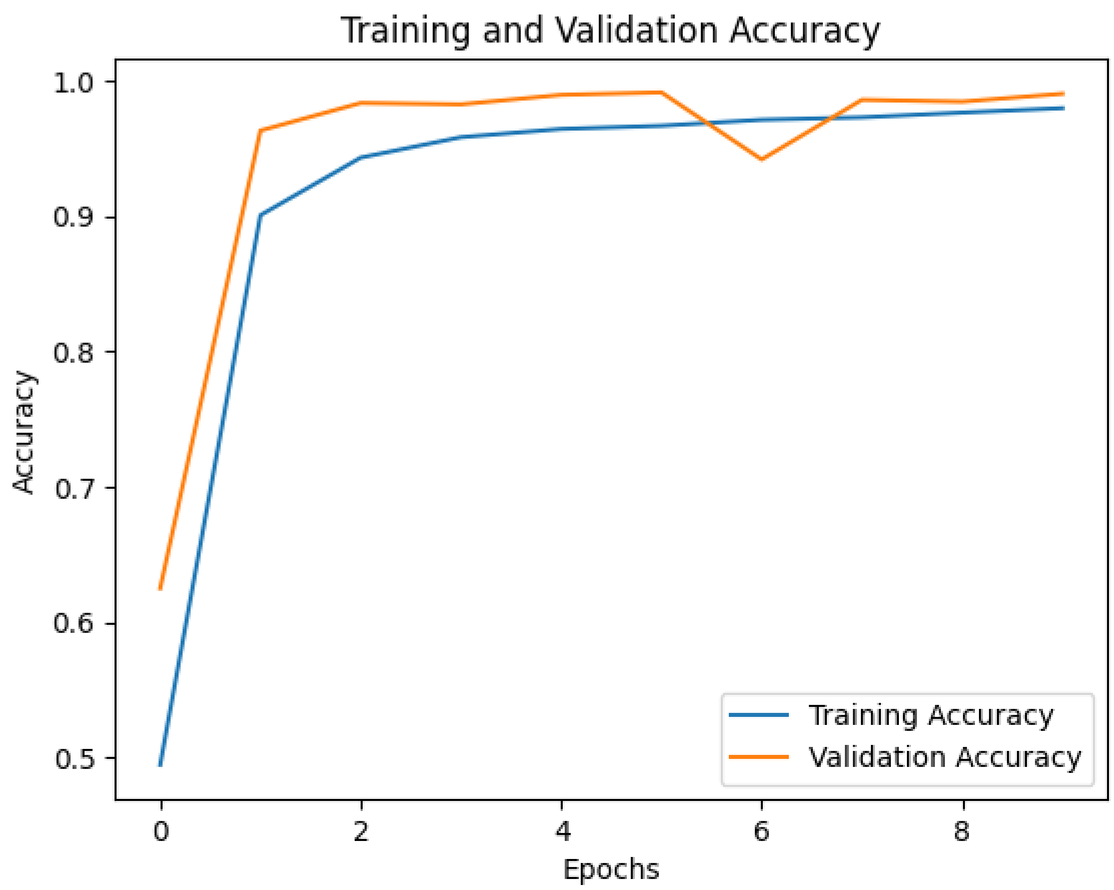


Different optimizers(Adam, RMSprop):

Adam results:

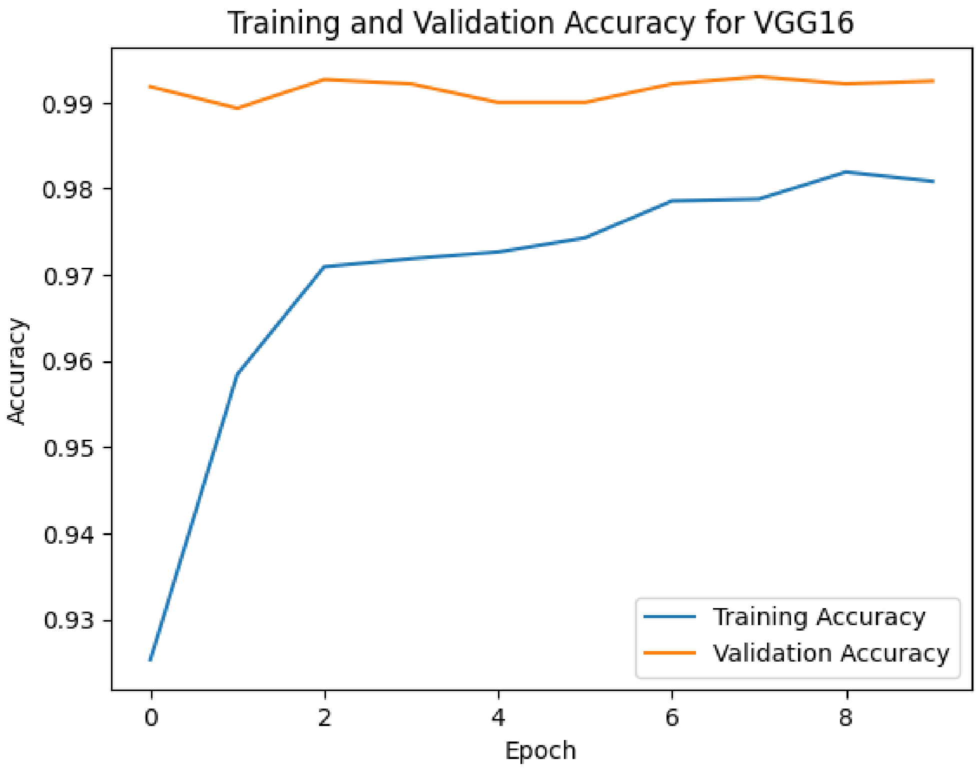


RMSprop results:



VGG16 results:

VGG16 Base Model stats:




```
Epoch 1/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 120s 240ms/step - accuracy: 0.8419 - loss: 2.7677 - val_accuracy: 0.9918 - val_loss: 0.0135
Epoch 2/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 97s 221ms/step - accuracy: 0.9669 - loss: 0.1669 - val_accuracy: 0.9893 - val_loss: 0.0130
Epoch 3/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 78s 178ms/step - accuracy: 0.9714 - loss: 0.1335 - val_accuracy: 0.9927 - val_loss: 0.0151
Epoch 4/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 82s 178ms/step - accuracy: 0.9712 - loss: 0.1739 - val_accuracy: 0.9922 - val_loss: 0.0122
Epoch 5/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 78s 178ms/step - accuracy: 0.9726 - loss: 0.1738 - val_accuracy: 0.9900 - val_loss: 0.0156
Epoch 6/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 78s 179ms/step - accuracy: 0.9763 - loss: 0.1495 - val_accuracy: 0.9900 - val_loss: 0.0156
Epoch 7/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 78s 178ms/step - accuracy: 0.9780 - loss: 0.1779 - val_accuracy: 0.9922 - val_loss: 0.0173
Epoch 8/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 82s 179ms/step - accuracy: 0.9783 - loss: 0.1334 - val_accuracy: 0.9930 - val_loss: 0.0121
Epoch 9/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 82s 178ms/step - accuracy: 0.9817 - loss: 0.1700 - val_accuracy: 0.9922 - val_loss: 0.0138
Epoch 10/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 100s 218ms/step - accuracy: 0.9806 - loss: 0.1614 - val_accuracy: 0.9925 - val_loss: 0.0123
188/188 - 23s - 122ms/step - accuracy: 0.9925 - loss: 0.0123
Validation accuracy: 0.9925
```

Different Learning Rates & results: [0.01, 0.001, 0.0001]

Learning rate = 0.01: The results for a learning rate of 0.01 show very poor performance. The model struggled to learn and remained stuck at an accuracy of around 52.78% for both training and validation. This indicates that the learning rate is too high, causing the model to overshoot.

Learning Rate = 0.001: The results for a learning rate of 0.001 shows a large improvement. The model achieves around 99.18% validation accuracy, which indicates it is learning, but the training and validation losses still show instability and overfitting.

Learning Rate = 0.0001: The results for a learning rate of 0.0001 is still very high at 99.05%. It is still slower and may require more epochs to achieve optimal performance.

Data augmentation results:

- Flip

Validation accuracy: 69%

Average training time: 100.5 sec

Total training time: 20.1 min

```
Epoch 1/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 91s 198ms/step - accuracy: 0.4242 - loss: 1.8140 - val_accuracy: 0.6742 - val_loss: 1.1491
Epoch 2/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 85s 193ms/step - accuracy: 0.7597 - loss: 0.6223 - val_accuracy: 0.6817 - val_loss: 1.1142
Epoch 3/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 86s 195ms/step - accuracy: 0.8053 - loss: 0.4866 - val_accuracy: 0.6757 - val_loss: 1.3318
Epoch 4/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 140s 191ms/step - accuracy: 0.8143 - loss: 0.4414 - val_accuracy: 0.6812 - val_loss: 1.3048
Epoch 5/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 143s 194ms/step - accuracy: 0.8218 - loss: 0.4310 - val_accuracy: 0.7157 - val_loss: 1.4692
Epoch 6/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 141s 192ms/step - accuracy: 0.8326 - loss: 0.4114 - val_accuracy: 0.6947 - val_loss: 1.2050
Epoch 7/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 143s 195ms/step - accuracy: 0.8297 - loss: 0.3993 - val_accuracy: 0.7183 - val_loss: 1.5031
Epoch 8/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 101s 230ms/step - accuracy: 0.8342 - loss: 0.3886 - val_accuracy: 0.6927 - val_loss: 1.4206
Epoch 9/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 126s 195ms/step - accuracy: 0.8348 - loss: 0.3934 - val_accuracy: 0.7055 - val_loss: 1.5935
Epoch 10/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 85s 193ms/step - accuracy: 0.8377 - loss: 0.3877 - val_accuracy: 0.6875 - val_loss: 1.6998
188/188 - 24s - 128ms/step - accuracy: 0.6900 - loss: 1.7073
Validation accuracy with the applied augmentation: 0.6900
```

- Rotation

Validation accuracy: 68%

Average training time:283.27 sec

Total training time: 51.93 min

```
438/438 ━━━━━━━━━━━━━━━━━━━━ 284s 634ms/step - accuracy: 0.4224 - loss: 1.7583 - val_accuracy: 0.5952 - val_loss: 1.2945
Epoch 2/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 277s 627ms/step - accuracy: 0.6962 - loss: 0.7769 - val_accuracy: 0.6905 - val_loss: 1.0954
Epoch 3/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 275s 623ms/step - accuracy: 0.7512 - loss: 0.6194 - val_accuracy: 0.6385 - val_loss: 1.2439
Epoch 4/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 277s 627ms/step - accuracy: 0.7469 - loss: 0.6196 - val_accuracy: 0.7273 - val_loss: 1.0786
Epoch 5/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 276s 622ms/step - accuracy: 0.7657 - loss: 0.5594 - val_accuracy: 0.6780 - val_loss: 1.3712
Epoch 6/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 325s 630ms/step - accuracy: 0.7716 - loss: 0.5411 - val_accuracy: 0.7407 - val_loss: 1.1986
Epoch 7/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 281s 636ms/step - accuracy: 0.7784 - loss: 0.5399 - val_accuracy: 0.7253 - val_loss: 1.4463
Epoch 8/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 341s 774ms/step - accuracy: 0.7764 - loss: 0.5352 - val_accuracy: 0.6882 - val_loss: 1.5396
Epoch 9/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 325s 644ms/step - accuracy: 0.7754 - loss: 0.5277 - val_accuracy: 0.7265 - val_loss: 1.8446
Epoch 10/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 279s 630ms/step - accuracy: 0.7940 - loss: 0.4995 - val_accuracy: 0.6800 - val_loss: 2.2289
188/188 - 86s - 457ms/step - accuracy: 0.6848 - loss: 2.2532
Validation accuracy with the applied augmentation: 0.6848
```

- Scaling

Validation accuracy: 69%

Average training time: 268.36 sec

Total training time: 49.2 min

```
Epoch 1/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 290s 651ms/step - accuracy: 0.5800 - loss: 1.4646 - val_accuracy: 0.7242 - val_loss: 0.9373
Epoch 2/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 282s 638ms/step - accuracy: 0.8994 - loss: 0.2892 - val_accuracy: 0.6950 - val_loss: 1.1845
Epoch 3/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 318s 629ms/step - accuracy: 0.9246 - loss: 0.2058 - val_accuracy: 0.7178 - val_loss: 1.3363
Epoch 4/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 383s 766ms/step - accuracy: 0.9281 - loss: 0.1799 - val_accuracy: 0.6700 - val_loss: 1.6364
Epoch 5/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 277s 627ms/step - accuracy: 0.9332 - loss: 0.1612 - val_accuracy: 0.7133 - val_loss: 1.4189
Epoch 6/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 384s 769ms/step - accuracy: 0.9313 - loss: 0.1617 - val_accuracy: 0.7078 - val_loss: 1.3585
Epoch 7/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 279s 629ms/step - accuracy: 0.9243 - loss: 0.1798 - val_accuracy: 0.6250 - val_loss: 2.0104
Epoch 8/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 320s 626ms/step - accuracy: 0.9178 - loss: 0.1934 - val_accuracy: 0.6415 - val_loss: 2.5024
Epoch 9/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 274s 619ms/step - accuracy: 0.9179 - loss: 0.1875 - val_accuracy: 0.6755 - val_loss: 2.4914
Epoch 10/10
438/438 ━━━━━━━━━━━━━━━━━━━━ 275s 618ms/step - accuracy: 0.8766 - loss: 0.2768 - val_accuracy: 0.6947 - val_loss: 2.2442
188/188 - 81s - 431ms/step - accuracy: 0.6975 - loss: 2.2107
Validation accuracy with the applied augmentation: 0.6975
```

Code snippet:

```
# 1. Rotation Only
data_augmentation = tf.keras.preprocessing.image.ImageDataGenerator(
    validation_split=0.3,  # Reserve 30% of the dataset for validation
    rotation_range=20,     # Rotate images by up to 20 degrees
    rescale=1./255         # Scale images to [0, 1] range
)

# 2. Flipping Only
# data_augmentation = tf.keras.preprocessing.image.ImageDataGenerator(
#     validation_split=0.3,  # Reserve 30% of the dataset for validation
#     horizontal_flip=True,  # Randomly flip images horizontally
#     rescale=1./255         # Scale images to [0, 1] range
# )

# 3. Scaling Only
# data_augmentation = tf.keras.preprocessing.image.ImageDataGenerator(
#     validation_split=0.3,  # Reserve 30% of the dataset for validation
#     zoom_range=0.2,        # Zoom in/out by up to 20%
#     rescale=1./255         # Scale images to [0, 1] range
# )
```

Batch size results:

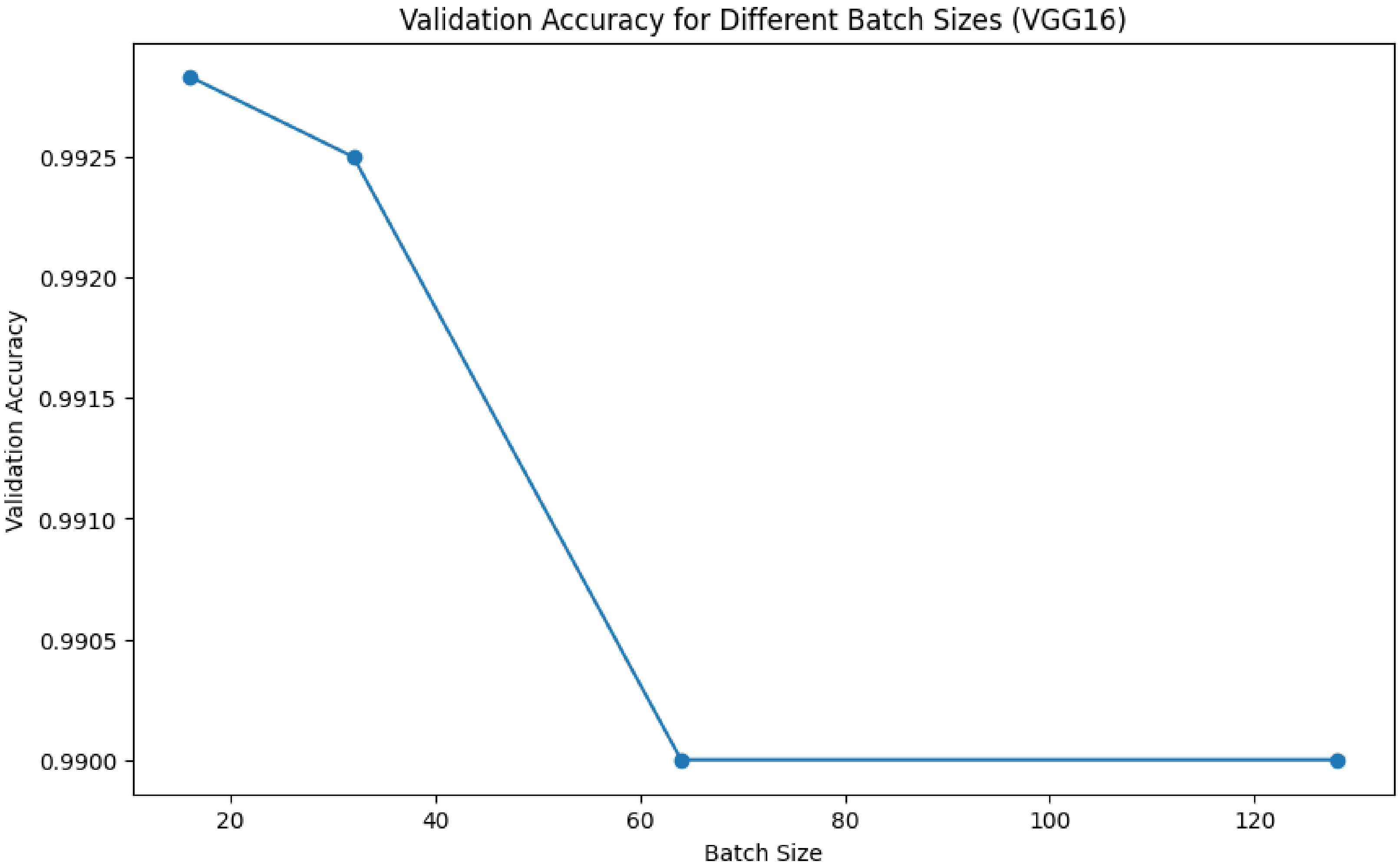
Tested sizes:

16 = 0.9928

32 = 0.9925

64 = 0.9900

128 = 0.9900

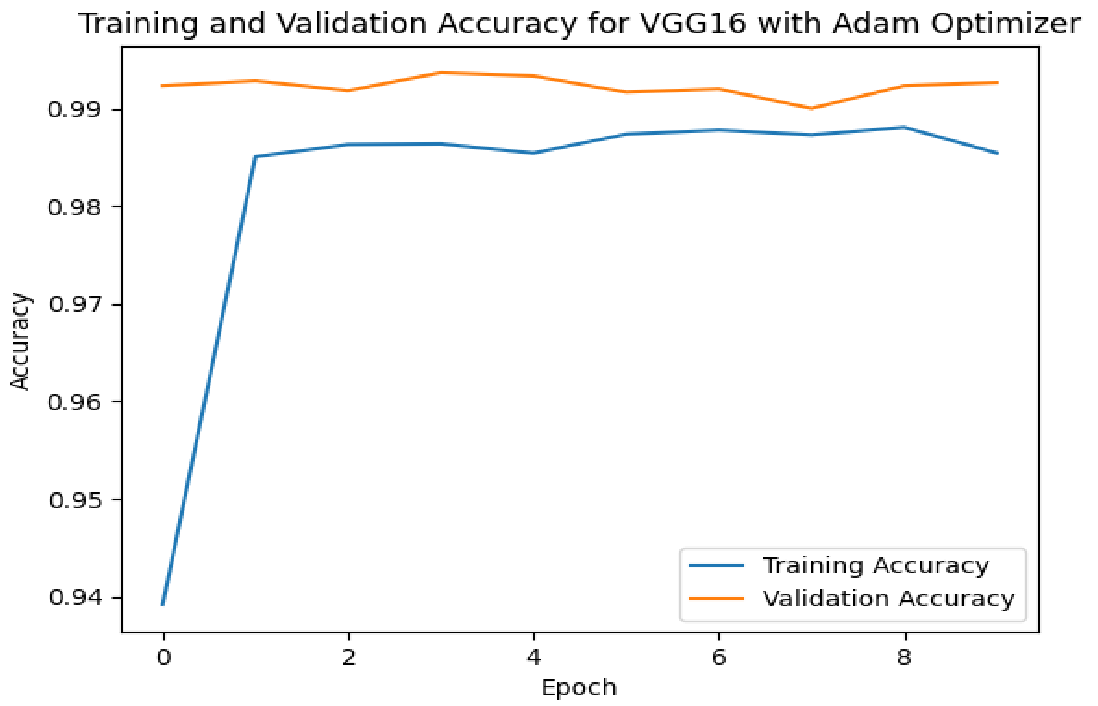


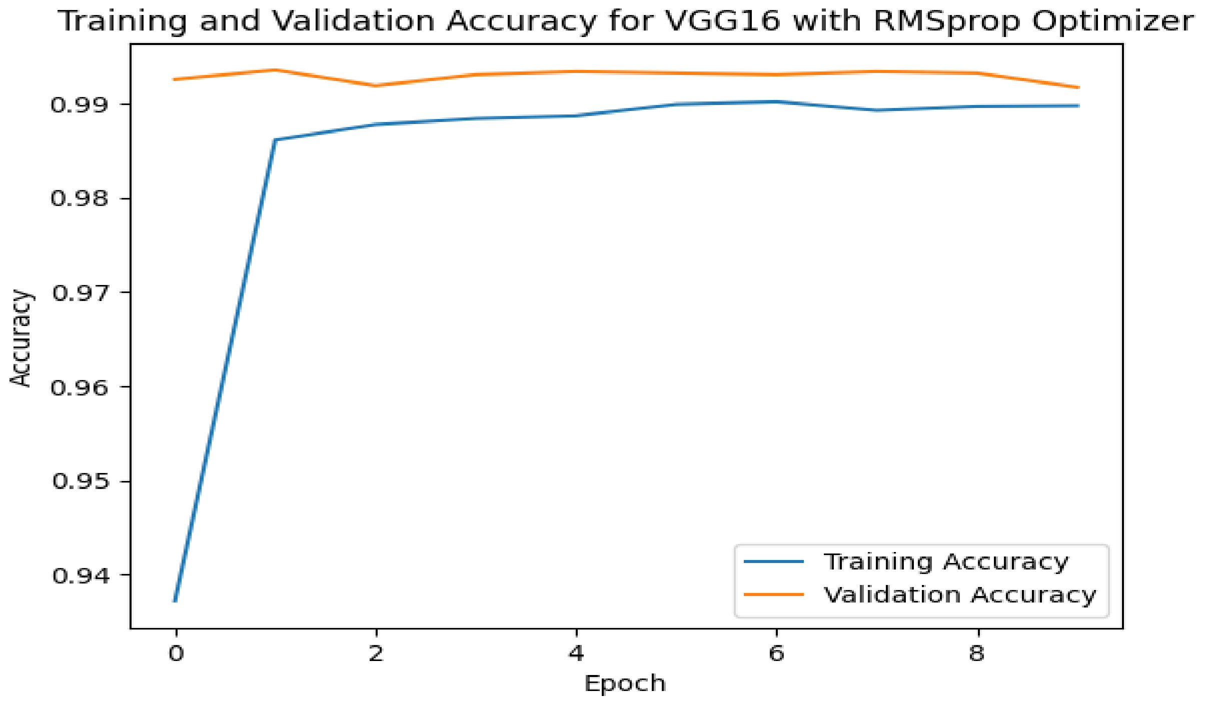
Different optimizers(Adam, RMSprop):

Adam results:

Accuracy:0.9927 %

Loss: 0.0177 %





RMSprop results:

Accuracy: 0.9917 %

Loss: 0.0187 %

Interpretation Of Data

AlexNet (base model):

Average accuracy: 0.9768
Loss: 0.0794
Average speed: 10.72 minutes
F1-score: 0.3

AlexNet optimal modifications for better performance:
Batch size: 16
Optimizer: RMSprop
Learning rate: 0.01
Data augmentation: rotation applied or auto augmentation.

VGG16 (base model):

Average accuracy: 0.9925
Loss: 0.0123
Average speed: 13.3 minutes
F1-score: 0.7

VGG16 optimal modifications for better performance:
Batch size: 16
Optimizer: Adam
Learning rate: 0.001
Data augmentation: flip or scaling both yield similar results of 69%.

Analysis Of Data

Optimized AlexNet Model Results:

Training time (using personal GPU): 50.3 min
Accuracy: 98.75 %
Improvement: 1.07 %

Optimized VGG16 Model results:

Training time (using A100 GPU's): 15min
Accuracy: 99.25
Improvement: N/A

CHAPTER FIVE

Conclusions and Recommendations

Conclusions

Impact of model architecture: AlexNet has a simple architecture, making it easy to train and it adapts well to small to medium-sized datasets. While VGG16 is better suited for very large datasets.

Learning rate & hyperparameter analysis: Smaller learning rates performed better in this round of testing; however, there was not enough appreciable evidence to conclude if one dramatically outperforms the other.

Impact of data augmentation: Comparing both models with data augmentation, the most successful techniques are scaling and rotation, which have the highest percent accuracy. However, it may depend on your specific model/task to determine which one is more appropriate.

Performance tradeoffs: The training times for both AlexNet and VGG16 on small to medium-sized datasets show that AlexNet trains significantly faster(when using the same GPU), while VGG16 requires more time due to its deeper architecture and additional convolutional layers.

Recommendations

1) If computation resources are limited, opt for ALexNet or a smaller variant of VGG16 (such as VGG with reduced convolutional layers).

2) Integrate batch normalization layers after the convolutional layers in both models to stabilize and speed up training and have consistent results.

3) For deployment in real-time applications or low-power devices, use AlexNet or even VGG16 to reduce its size without loosing too much performance. Because VGG16's large size can be a bottleneck for real-time applications.

4) Experiment with hybrid models that combine the strength of both architectures, such as using the first few layers of AlexNet for feature extraction and fully connected layers of VGG16 for classification.