המחלקה להנדסת חשמל ואלקטרוניקה

(31245) מערכות לומדות ולמידה עמוקה

Lab 8 report

פרנסיס עבוד

Prepared for: Dr. Amer Adler

Date: 13/06/2025

EX.1:

Load the following example into Colab: “Transfer learning with a pretrained ConvNet”: https://www.tensorflow.org/tutorials/images/transfer_learning

Code:

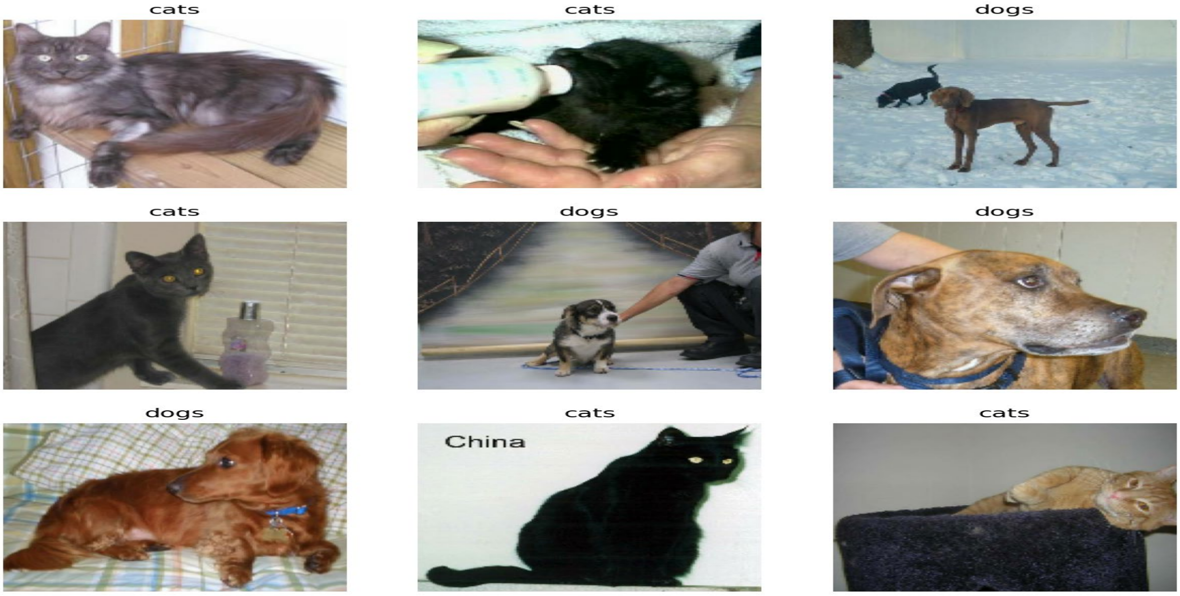
```python
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf

# ========================
# SECTION 1: Data Loading and Preprocessing (from TensorFlow tutorial)
# ========================

# Download and setup dataset
_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
path_to_zip = tf.keras.utils.get_file('cats_and_dogs.zip', origin=_URL, extract=True)
PATH = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_extracted', 'cats_and_dogs_filtered')
train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')

BATCH_SIZE = 32
IMG_SIZE = (160, 160)

# Create datasets
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    shuffle=True,
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE
)

validation_dataset = tf.keras.utils.image_dataset_from_directory(
    validation_dir,
    shuffle=True,
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE
)

class_names = train_dataset.class_names

# Visualize the data
plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
plt.savefig('images/dataset_samples.png', dpi=150, bbox_inches='tight')
plt.close()  # Close the figure to free memory
print("Dataset samples saved to images/dataset_samples.png")
```

```python
# Split validation dataset into validation and test
val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)

print('Number of validation batches: %d' % tf.data.experimental.cardinality(validation_dataset))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_dataset))

# Configure dataset for performance
AUTOTUNE = tf.data.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)

# Data augmentation
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2),
])

# Visualize augmented images
for image, _ in train_dataset.take(1):
    plt.figure(figsize=(10, 10))
    first_image = image[0]
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        augmented_image = data_augmentation(tf.expand_dims(first_image, 0))
        plt.imshow(augmented_image[0] / 255)
        plt.axis('off')
plt.savefig('images/data_augmentation.png', dpi=150, bbox_inches='tight')
plt.close()  # Close the figure to free memory
print("Data augmentation visualization saved to images/data_augmentation.png")

# Setup preprocessing and base model
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input

# Create the base model from the pre-trained model MobileNetV2
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')

# Freeze the base model (154 layers)
base_model.trainable = False
```

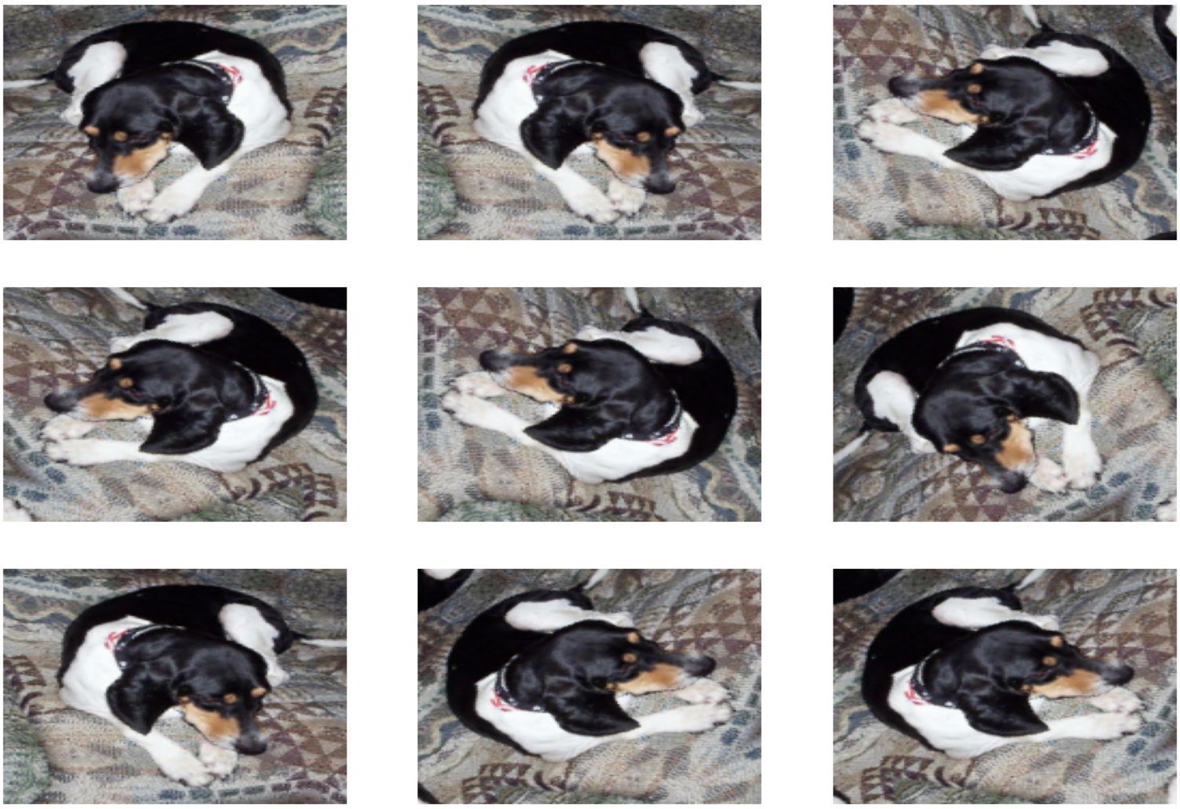
```python

# Add classification layers
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
prediction_layer = tf.keras.layers.Dense(1)

# Build the complete model
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)

print(f"Total layers in base model: {len(base_model.layers)}")
```

Output:

- ### Dataset sample:

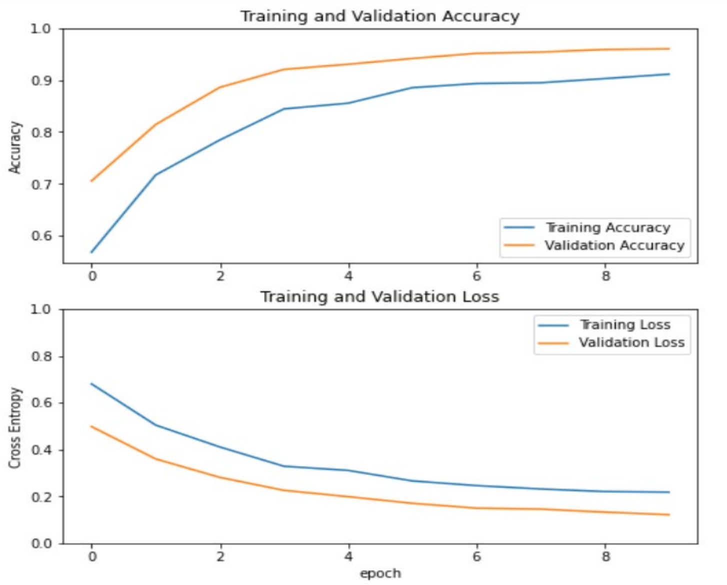


- ### Data augmentation:



Ex.2:

Freeze the base CNN layers (MobileNetV2 with 154 layers), and train only the top layers. Provide a loss graph and accuracy graph (vs. epoch number) for 3 different learning rates: 0.01, 0.001, 0.0001 and using SGD, AdaGrad, Adam and RMSprop optimizers. Which learning rate & optimizer provided the best test-set accuracy results?



Code:

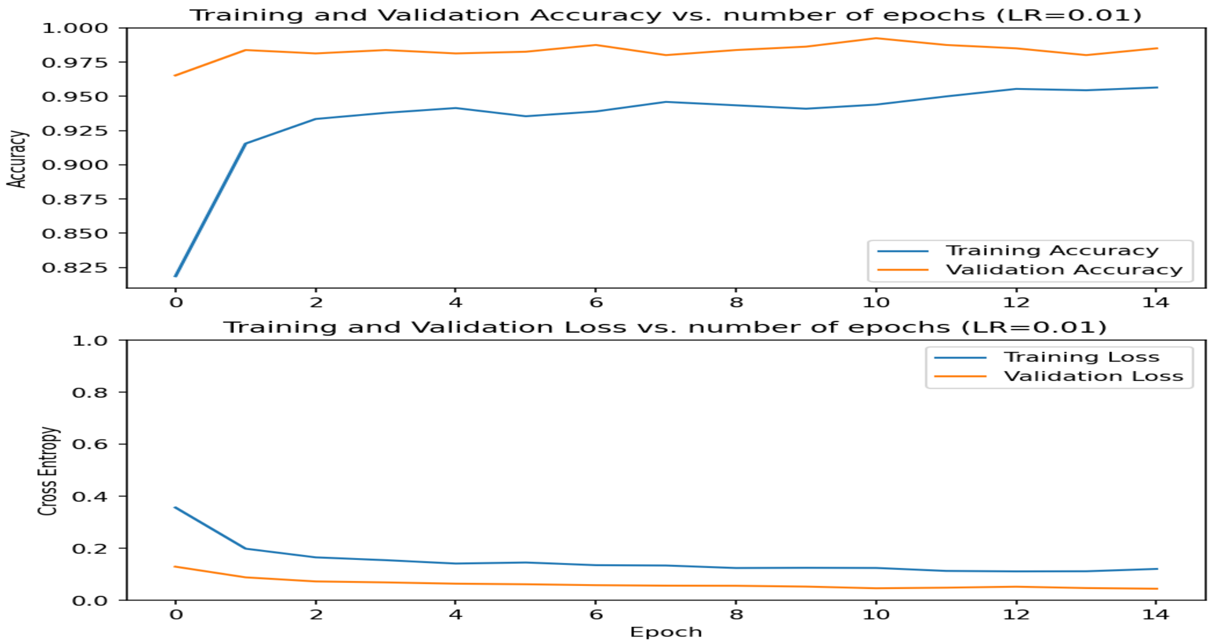
```python
# ==========================
# SECTION 2: Feature Extraction with Different Optimizers and Learning Rates
# Freeze base CNN layers and train only top layers
# Test: SGD, AdaGrad, Adam, RMSprop with learning rates 0.01, 0.001, 0.0001
# ==========================

optimizers = {
    'SGD': tf.keras.optimizers.SGD,
    'AdaGrad': tf.keras.optimizers.Adagrad,
    'Adam': tf.keras.optimizers.Adam,
    'RMSprop': tf.keras.optimizers.RMSprop
}

learning_rates = [0.01, 0.001, 0.0001]
results = {}
best_accuracy = 0
best_optimizer = None
best_lr = None

print("=" * 80)
print("SECTION 2: FEATURE EXTRACTION (BASE MODEL FROZEN)")
print("=" * 80)

for optimizer_name, optimizer_class in optimizers.items():
    for LearnRate in learning_rates:
        print(f"\nTraining with {optimizer_name} optimizer and learning rate {LearnRate}")
        print("-" * 60)

        # Compile the model
        model.compile(optimizer=optimizer_class(learning_rate=LearnRate),
                      loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                      metrics=['accuracy'])

        initial_epochs = 10

        # Evaluate before training
        loss0, accuracy0 = model.evaluate(validation_dataset, verbose=0)
        print(f"Initial loss: {loss0:.4f}, Initial accuracy: {accuracy0:.4f}")

        # Train the model
        history = model.fit(train_dataset,
                            epochs=15,
                            validation_data=validation_dataset,
                            verbose=1)

        # Evaluate on test dataset
        test_loss, test_accuracy = model.evaluate(test_dataset, verbose=0)
        print(f"Test accuracy: {test_accuracy:.4f}")

        # Store results
        key = f"{optimizer_name}_lr_{LearnRate}"
        results[key] = {
            'history': history,
            'test_accuracy': test_accuracy,
            'optimizer': optimizer_name,
            'learning_rate': LearnRate
        }
```

```python
        # Track best combination
        if test_accuracy > best_accuracy:
            best_accuracy = test_accuracy
            best_optimizer = optimizer_name
            best_lr = LearnRate

        # Extract training history
        acc = history.history['accuracy']
        val_acc = history.history['val_accuracy']
        loss = history.history['loss']
        val_loss = history.history['val_loss']

        # Plot training results
        plt.figure(figsize=(8, 8))
        plt.subplot(2, 1, 1)
        plt.plot(acc, label='Training Accuracy')
        plt.plot(val_acc, label='Validation Accuracy')
        plt.ylabel('Accuracy')
        plt.ylim([min(plt.ylim()),1])
        plt.title(f"Training and Validation Accuracy vs. number of epochs (LR={LearnRate})")
        plt.legend(loc='lower right')

        plt.subplot(2, 1, 2)
        plt.plot(loss, label='Training Loss')
        plt.plot(val_loss, label='Validation Loss')
        plt.xlabel('Epoch')
        plt.ylabel('Cross Entropy')
        plt.ylim([0,1.0])
        plt.title(f"Training and Validation Loss vs. number of epochs (LR={LearnRate})")
        plt.legend()

        # Save the plot with descriptive filename
        filename = f'images/training_{optimizer_name}_lr{LearnRate}.png'
        plt.savefig(filename, dpi=150, bbox_inches='tight')
        plt.close()  # Close the figure to free memory
        print(f"Training plot saved to {filename}")

# Print Section 2 results
print("\n" + "=" * 80)
print("SECTION 2 RESULTS COMPARISON")
print("=" * 80)
print(f"{'Optimizer':<12} {'Learning Rate':<15} {'Test Accuracy':<15}")
print("-" * 45)

for key, result in results.items():
    optimizer = result['optimizer']
    lr = result['learning_rate']
    test_acc = result['test_accuracy']
    print(f"{optimizer:<12} {lr:<15} {test_acc:<15.4f}")

print(f"\nBest combination: {best_optimizer} with learning rate {best_lr}")
print(f"Best test accuracy: {best_accuracy:.4f}")
```

Output:

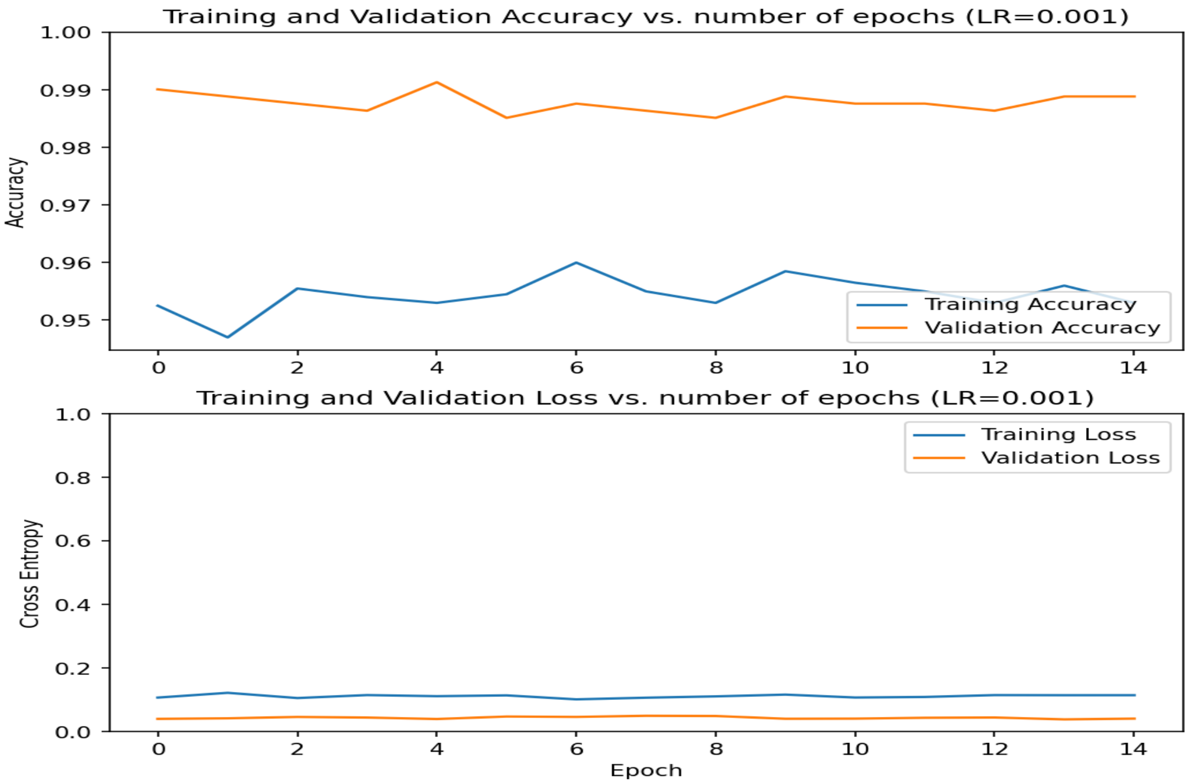
- **SGD:**
  - **Learning rate set to 0.01:**

```
Test accuracy: 0.9635
```



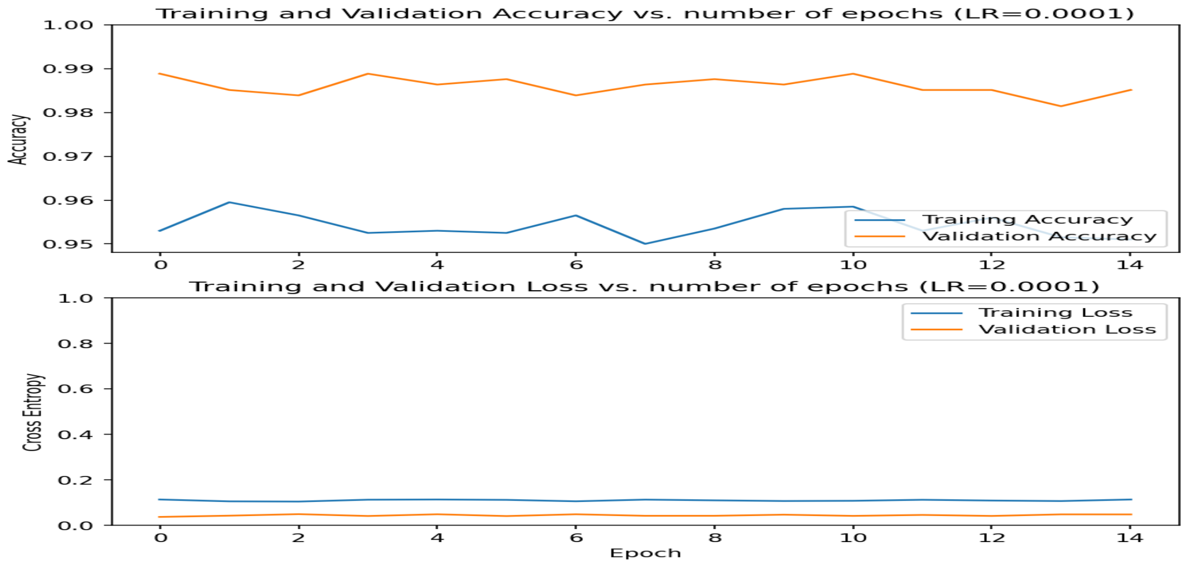
  - **Learning rate set to 0.001:**

```
Test accuracy: 0.9688
```



- **Learning rate set to 0.0001:**

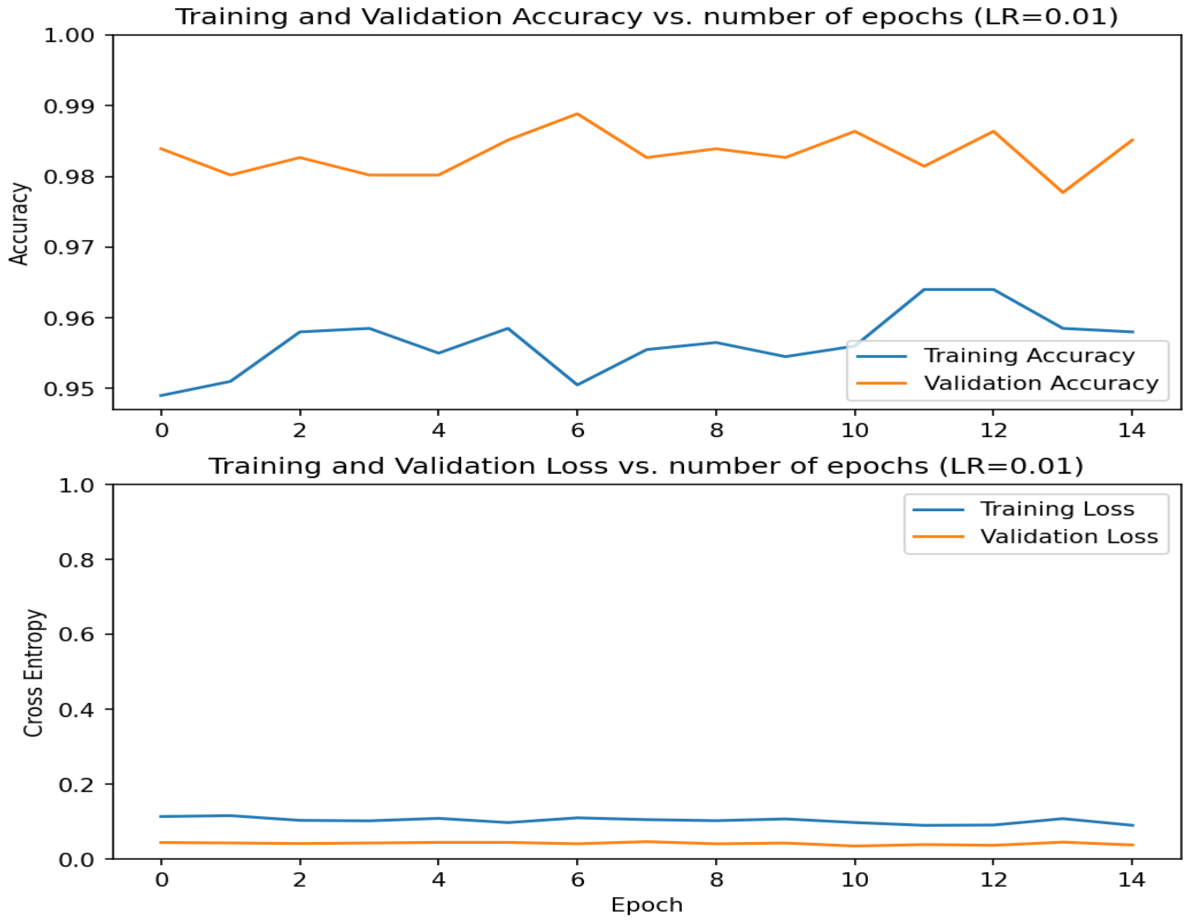
Test accuracy: 0.9740



➢ <u>**AdaGrad:**</u>

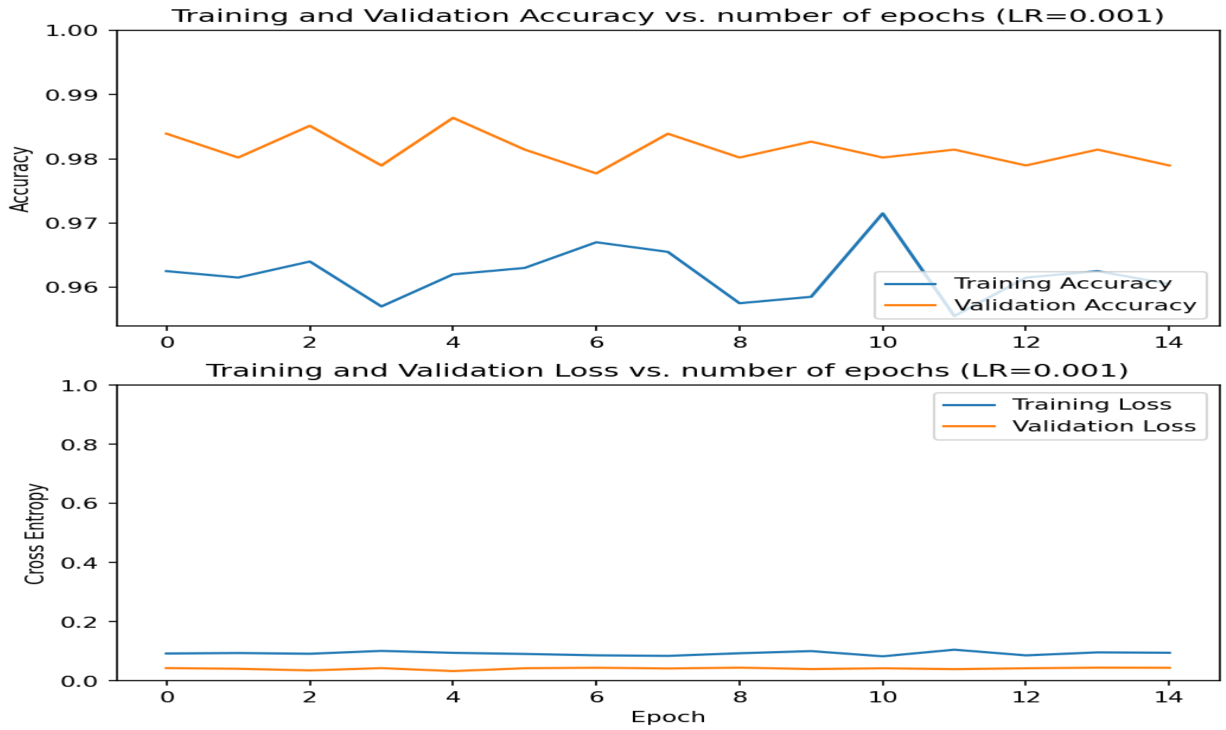
- **Learning rate set to 0.01:**

Test accuracy: 0.9792



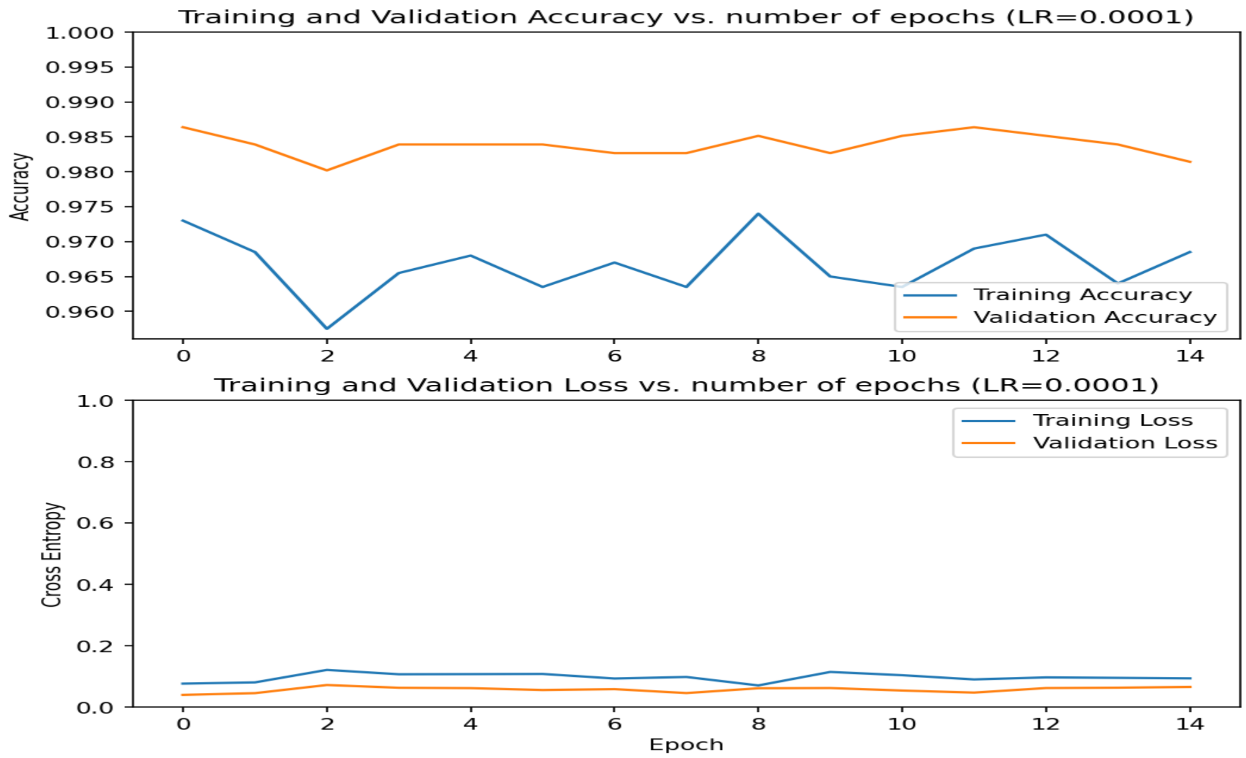
- **Learning rate set to 0.001:**

```
Test accuracy: 0.9635
```



- **Learning rate set to 0.0001:**

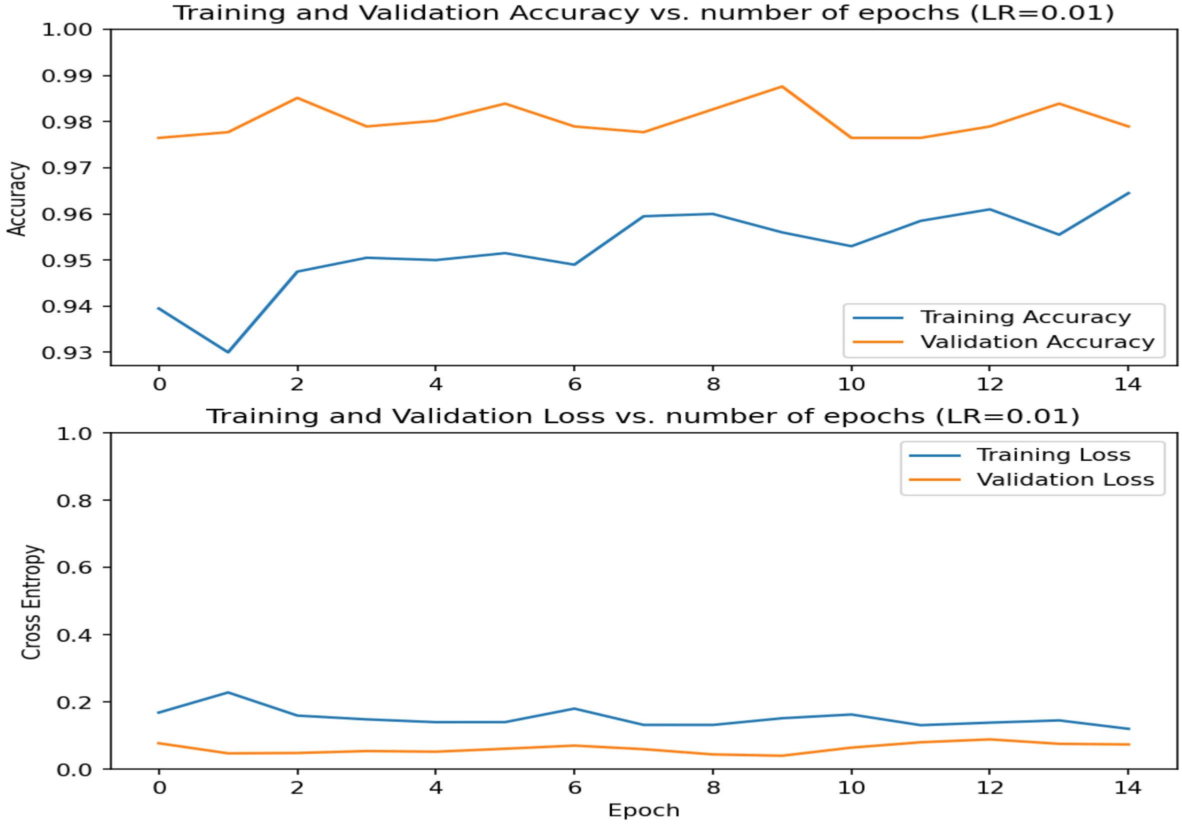
```
Test accuracy: 0.9635
```



➢ Adam:

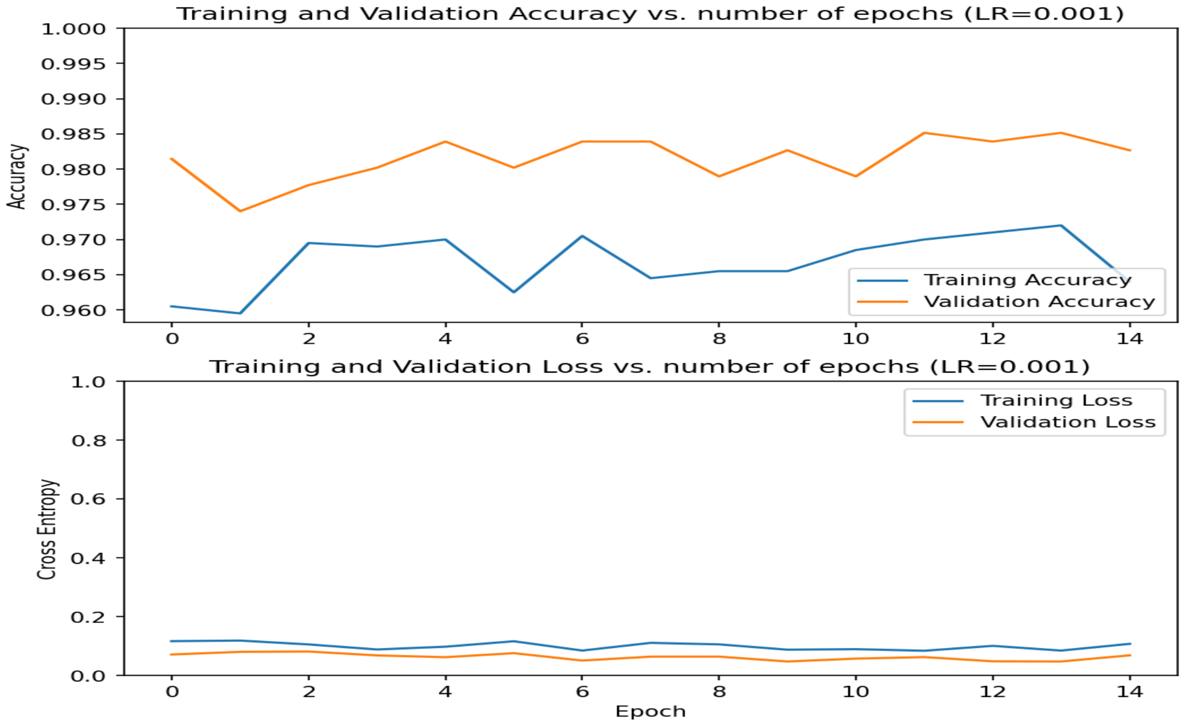
- **Learning rate set to 0.01:**

```
Test accuracy: 0.9688
```



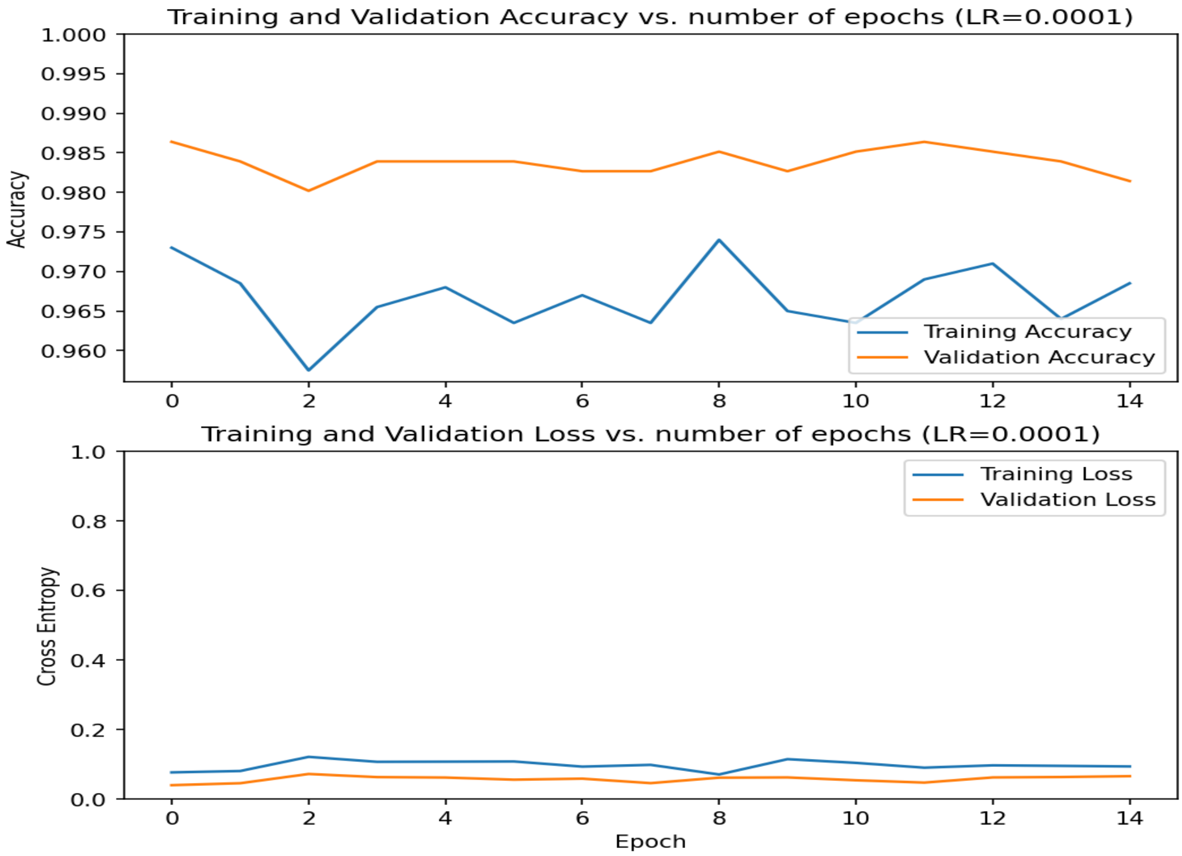
- **Learning rate set to 0.001:**

```
Test accuracy: 0.9740
```



- **Learning rate set to 0.0001:**

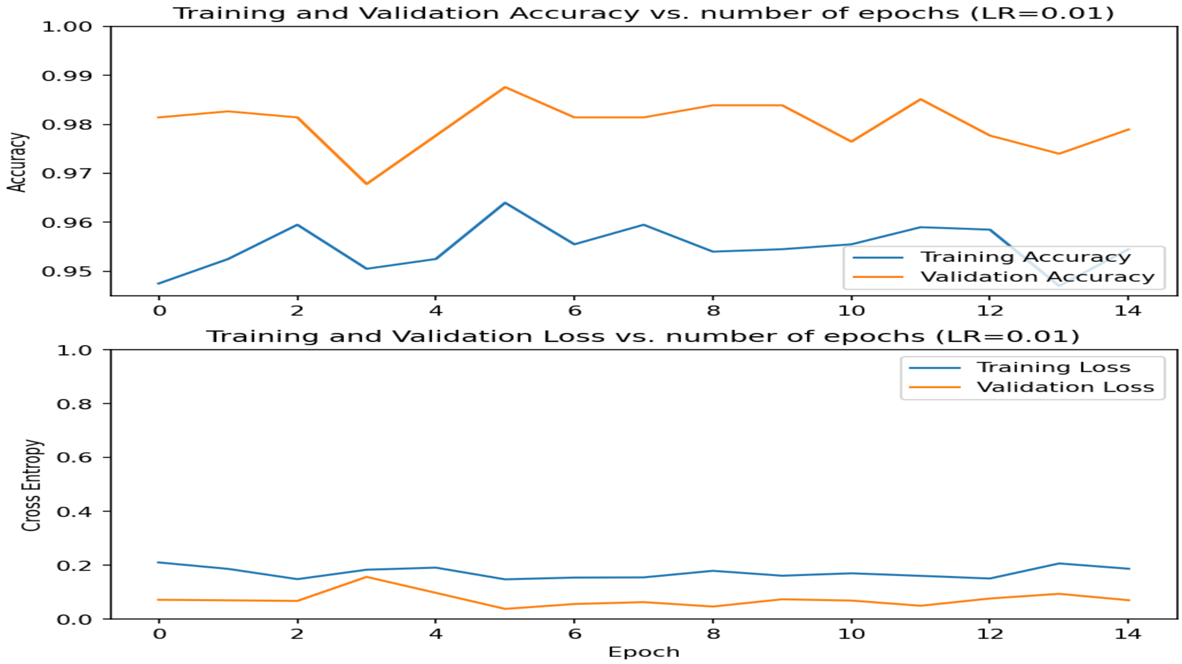
Test accuracy: 0.9844



➢ RMSprop:

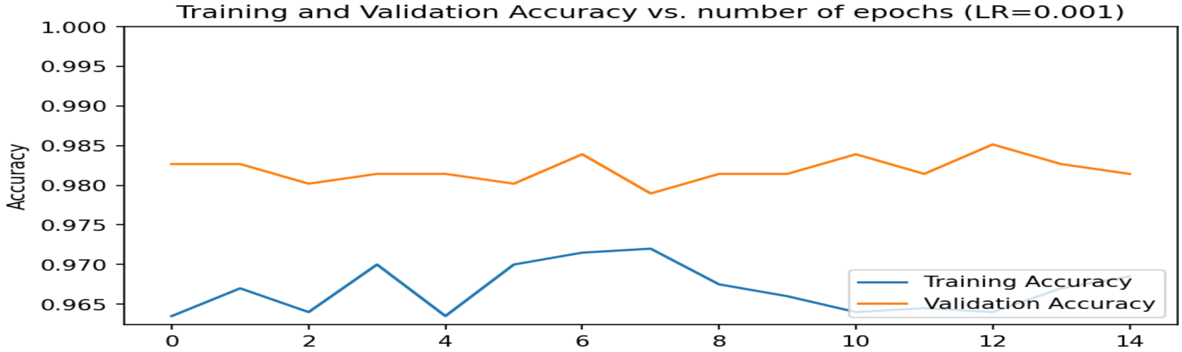
- **Learning rate set to 0.01:**

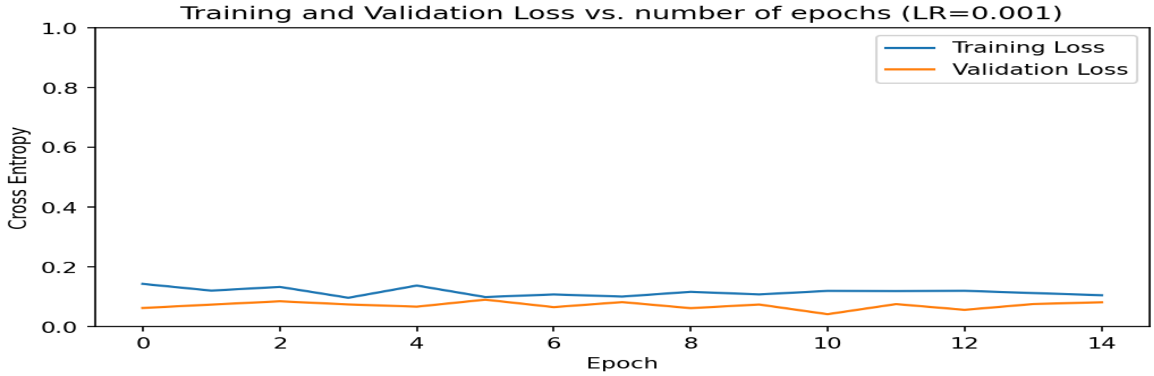
Test accuracy: 0.9635



- **Learning rate set to 0.001:**

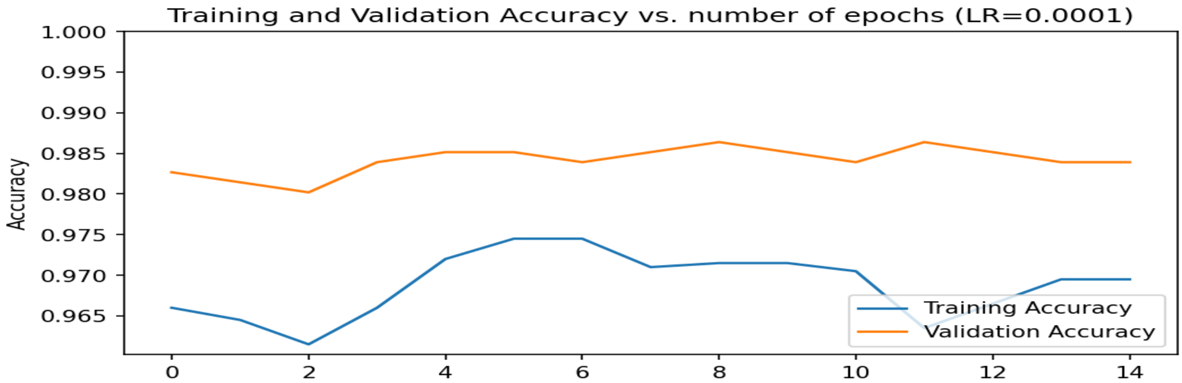
`Test accuracy: 0.9740`

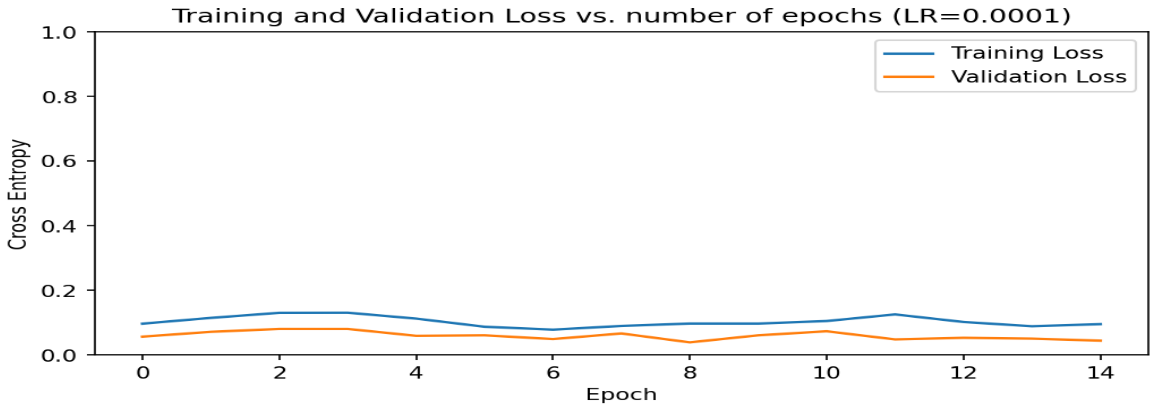




- **Learning rate set to 0.0001:**

`Test accuracy: 0.9740`





RESULTS COMPARISON

```
==================================================
SECTION 2 RESULTS COMPARISON
==================================================
Optimizer      Learning Rate    Test Accuracy
---------------------------------------------
SGD            0.01             0.9635
SGD            0.001            0.9688
SGD            0.0001           0.9740
AdaGrad        0.01             0.9792
AdaGrad        0.001            0.9635
AdaGrad        0.0001           0.9635
Adam           0.01             0.9688
Adam           0.001            0.9740
Adam           0.0001           0.9844
RMSprop        0.01             0.9635
RMSprop        0.001            0.9740
RMSprop        0.0001           0.9740

Best combination: Adam with learning rate 0.0001
Best test accuracy: 0.9844
```

Ex.3:

Perform fine-tuning of the base network (starting from layer 100, while layers below 100 are frozen) and the inference layers, using the best optimizer from section 2. Use the learning rates from section 2, after division by 10. Which learning rate provided the best test-set accuracy results?

Code:

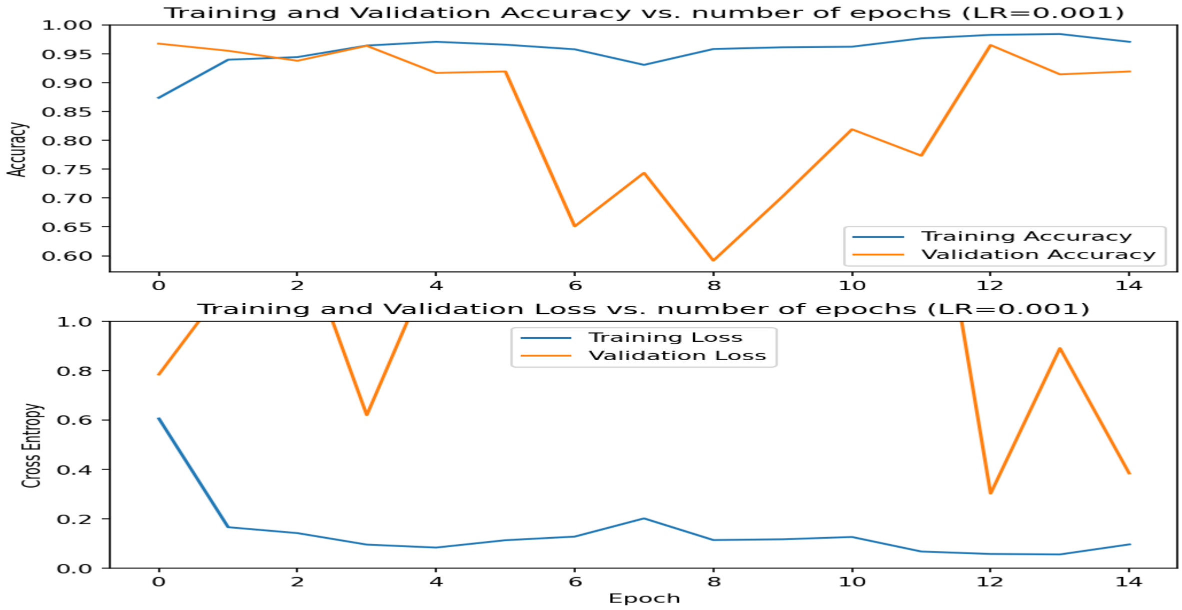
```python
# ========================
# SECTION 3: Fine-Tuning
# Unfreeze from layer 100, keep layers below 100 frozen
# Use best optimizer from Section 2
# Use learning rates from Section 2 divided by 10: 0.001, 0.0001, 0.00001
# ========================

print("\n" + "=" * 80)
print("SECTION 3: FINE-TUNING (UNFREEZE FROM LAYER 100)")
print("=" * 80)

# Unfreeze the base model
base_model.trainable = True

# Freeze layers below 100
for layer in base_model.layers[:100]:
    layer.trainable = False

print(f"Number of layers in the base model: {len(base_model.layers)}")
print(f"Fine-tuning from layer 100 onwards")

# Learning rates for fine-tuning (divided by 10)
fine_tune_learning_rates = [lr / 10 for lr in learning_rates]  # [0.001, 0.0001, 0.00001]

fine_tune_results = {}
best_fine_tune_accuracy = 0
best_fine_tune_lr = None

# Use the best optimizer from Section 2
best_optimizer_class = optimizers[best_optimizer]

for LearnRate in fine_tune_learning_rates:
    print(f"\nFine-tuning with {best_optimizer} optimizer and learning rate {LearnRate}")
    print("-" * 60)

    # Compile the model with fine-tuning learning rate
    model.compile(optimizer=best_optimizer_class(learning_rate=LearnRate),
                  loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                  metrics=['accuracy'])

    initial_epochs = 10

    # Evaluate before fine-tuning
    loss0, accuracy0 = model.evaluate(validation_dataset, verbose=0)
    print(f"Initial loss: {loss0:.4f}, Initial accuracy: {accuracy0:.4f}")

    # Fine-tune the model
    history = model.fit(train_dataset,
                        epochs=15,
                        validation_data=validation_dataset,
                        verbose=1)

    # Evaluate on test dataset
    test_loss, test_accuracy = model.evaluate(test_dataset, verbose=0)
    print(f"Fine-tuned test accuracy: {test_accuracy:.4f}")

    # Store fine-tuning results
    fine_tune_results[LearnRate] = {
        'history': history,
        'test_accuracy': test_accuracy
    }

    # Track best fine-tuning learning rate
    if test_accuracy > best_fine_tune_accuracy:
        best_fine_tune_accuracy = test_accuracy
        best_fine_tune_lr = LearnRate
      # Extract training history
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    # Plot fine-tuning results
    plt.figure(figsize=(8, 8))
    plt.subplot(2, 1, 1)
    plt.plot(acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.ylabel('Accuracy')
    plt.ylim([min(plt.ylim()),1])
    plt.title(f"Training and Validation Accuracy vs. number of epochs (LR={LearnRate})")
    plt.legend(loc='lower right')

    plt.subplot(2, 1, 2)
    plt.plot(loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Cross Entropy')
    plt.ylim([0,1.0])
    plt.title(f"Training and Validation Loss vs. number of epochs (LR={LearnRate})")
    plt.legend()

    # Save the fine-tuning plot
    filename = f'images/fine_tuning_lr{LearnRate}.png'
    plt.savefig(filename, dpi=150, bbox_inches='tight')
    plt.close()  # Close the figure to free memory
    print(f"Fine-tuning plot saved to {filename}")

# Print Section 3 results
print("\n" + "=" * 80)
print("SECTION 3 FINE-TUNING RESULTS")
print("=" * 80)
print(f"{'Learning Rate':<15} {'Test Accuracy':<15}")
print("-" * 30)

for lr, result in fine_tune_results.items():
    test_acc = result['test_accuracy']
    print(f"{lr:<15} {test_acc:<15.4f}")

print(f"\nBest fine-tuning learning rate: {best_fine_tune_lr}")
print(f"Best fine-tuning test accuracy: {best_fine_tune_accuracy:.4f}")
print(f"Improvement from feature extraction: {best_fine_tune_accuracy - best_accuracy:.4f}")
```

Output:

```
==============================================================
SECTION 3: FINE-TUNING (UNFREEZE FROM LAYER 100)
==============================================================
Number of layers in the base model: 154
Fine-tuning from layer 100 onwards

Fine-tuning with Adam optimizer and learning rate 0.001
```

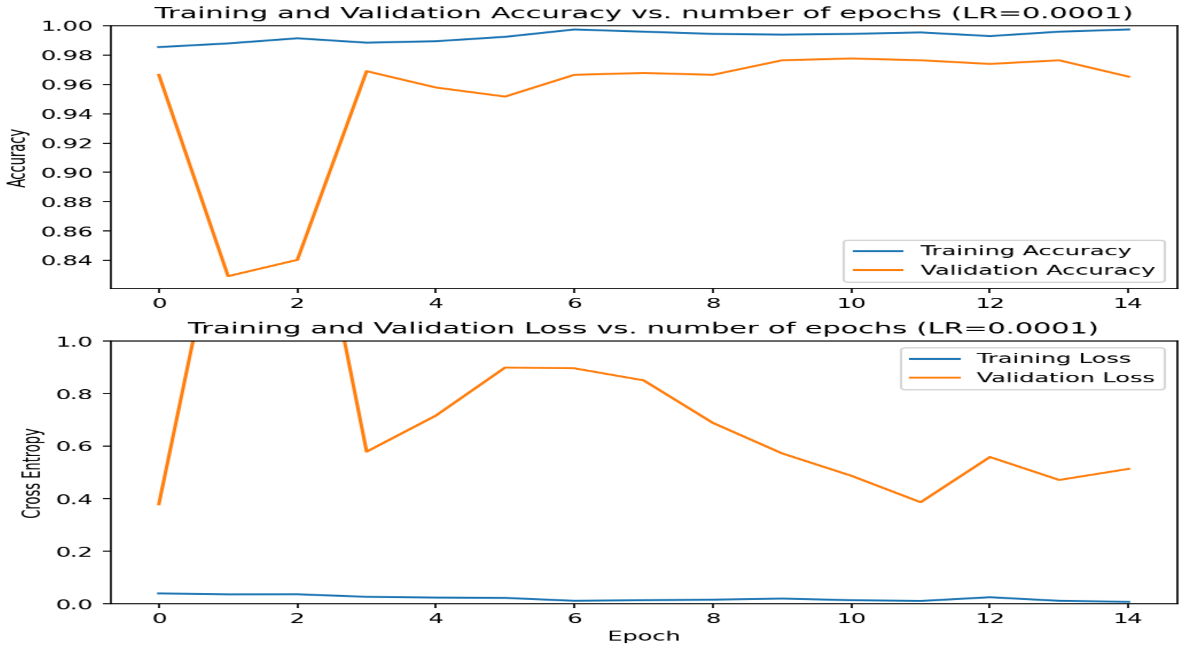
- **Learning rate set to 0.001:**

```
Fine-tuned test accuracy: 0.9115
```



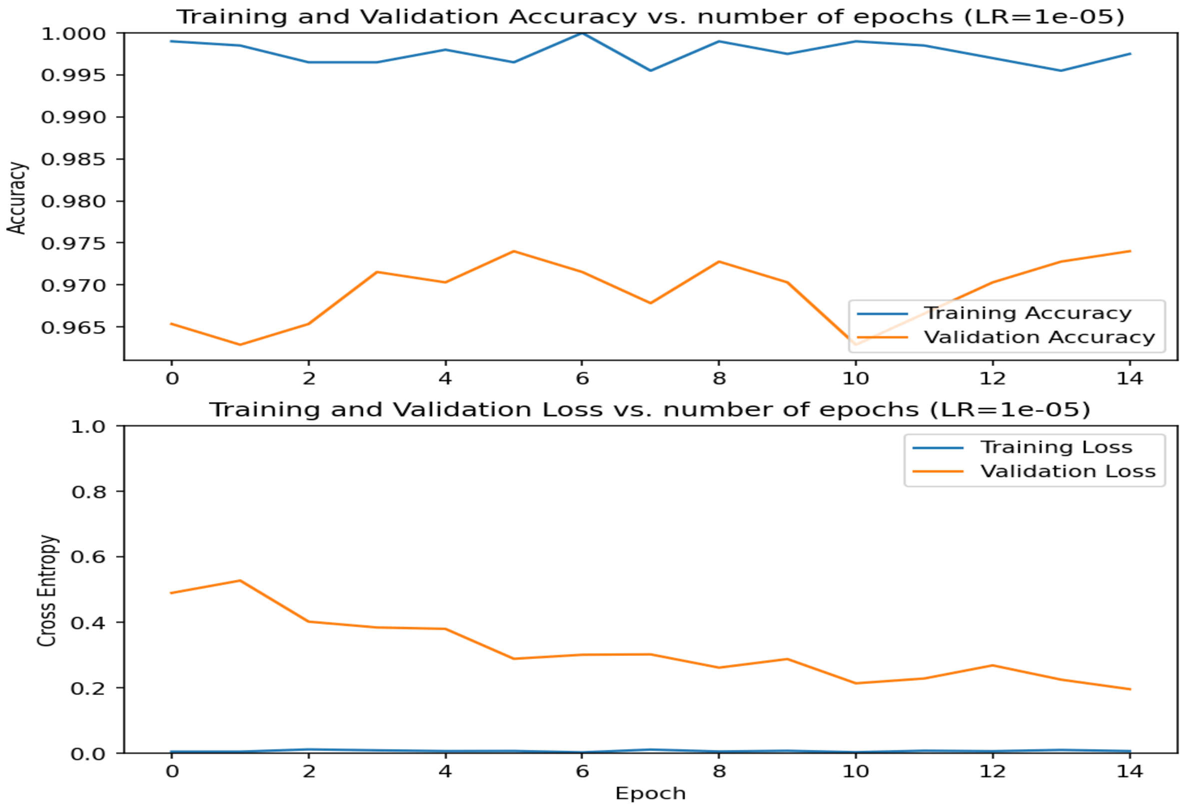
- **Learning rate set to 0.0001:**

```
Fine-tuned test accuracy: 0.9583
```



- **Learning rate set to 0.00001:**

```
Fine-tuned test accuracy: 0.9688
```



FINE-TUNING RESULTS

```
============================================================
SECTION 3 FINE-TUNING RESULTS
============================================================
Learning Rate    Test Accuracy
------------------------------
0.001            0.9115
0.0001           0.9583
1e-05            0.9688

Best fine-tuning learning rate: 1e-05
Best fine-tuning test accuracy: 0.9688
Improvement from feature extraction: -0.0156
```

FINAL SUMMARY

```
FINAL SUMMARY
==========================================
SECTION 2 (Feature Extraction):
  Best optimizer: Adam
  Best learning rate: 0.0001
  Best test accuracy: 0.9844

SECTION 3 (Fine-tuning):
  Best optimizer: Adam (from Section 2)
  Best test accuracy: 0.9844

SECTION 3 (Fine-tuning):
  Best optimizer: Adam (from Section 2)
SECTION 3 (Fine-tuning):
  Best optimizer: Adam (from Section 2)
  Best optimizer: Adam (from Section 2)
  Best learning rate: 1e-05
  Best test accuracy: 0.9688
  Total improvement: -0.0156
```