Code Appendix

Name: Mark Ma UNI: km4054

Nov 7, 2024

[17]:
```
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

1 Problem 1

[18]:
```
# Load and preprocess CIFAR-10 data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize the pixel values to be between 0 and 1
x_train, x_test = x_train.astype('float32') / 255.0, x_test.astype('float32') /␣
↪255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Define the original CNN model
def create_cnn_model():
    model = models.Sequential()

    # First convolutional block
    model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same',␣
↪input_shape=(32, 32, 3)))
    model.add(layers.MaxPooling2D((2, 2)))

    # Second convolutional block
    model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Third convolutional block
    model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Flatten and add dense layers
```

```
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(64, activation='relu'))

    # Output layer
    model.add(layers.Dense(10, activation='softmax'))  # 10 classes for CIFAR-10

    return model

# Define the modified model with an additional convolutional layer
def create_modified_cnn_model():
    model = models.Sequential()

    # First convolutional block
    model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same',␣
 ↪input_shape=(32, 32, 3)))
    model.add(layers.MaxPooling2D((2, 2)))

    # Second convolutional block
    model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Third convolutional block
    model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Additional convolutional block (added layer)
    model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Flatten and add dense layers
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(64, activation='relu'))

    # Output layer
    model.add(layers.Dense(10, activation='softmax'))  # 10 classes for CIFAR-10

    return model
```

[19]:
```
# Instantiate both models
original_model = create_cnn_model()
modified_model = create_modified_cnn_model()

# Compile both models
original_model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

```
modified_model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

# Print model summaries to compare parameter counts
print("Original Model Summary:")
original_model.summary()
```

```
Original Model Summary:
```

Model: "sequential_4"

```
 Layer (type)                     Output Shape                              ␣
↪Param #

 conv2d_13 (Conv2D)               (None, 32, 32, 32)                          ␣
↪896

 max_pooling2d_13 (MaxPooling2D)  (None, 16, 16, 32)                          ␣
↪  0

 conv2d_14 (Conv2D)               (None, 16, 16, 64)                       ␣
↪18,496

 max_pooling2d_14 (MaxPooling2D)  (None, 8, 8, 64)                            ␣
↪  0

 conv2d_15 (Conv2D)               (None, 8, 8, 128)                        ␣
↪73,856

 max_pooling2d_15 (MaxPooling2D)  (None, 4, 4, 128)                           ␣
↪  0

 flatten_4 (Flatten)              (None, 2048)                                ␣
↪  0

 dense_12 (Dense)                 (None, 128)                             ␣
↪262,272

 dense_13 (Dense)                 (None, 64)                                ␣
↪8,256

 dense_14 (Dense)                 (None, 10)                                  ␣
↪650
```

```
Total params: 364,426 (1.39 MB)

Trainable params: 364,426 (1.39 MB)

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

[20]:
```
print("\nModified Model Summary (with extra layer):")
modified_model.summary()
```

```
Modified Model Summary (with extra layer):

Model: "sequential_5"


 Layer (type)                       Output Shape                      ␣
↪Param #

 conv2d_16 (Conv2D)                 (None, 32, 32, 32)                  ␣
↪896

 max_pooling2d_16 (MaxPooling2D)    (None, 16, 16, 32)                  ␣
↪  0

 conv2d_17 (Conv2D)                 (None, 16, 16, 64)                ␣
↪18,496

 max_pooling2d_17 (MaxPooling2D)    (None, 8, 8, 64)                    ␣
↪  0

 conv2d_18 (Conv2D)                 (None, 8, 8, 128)                 ␣
↪73,856

 max_pooling2d_18 (MaxPooling2D)    (None, 4, 4, 128)                   ␣
↪  0

 conv2d_19 (Conv2D)                 (None, 4, 4, 256)                ␣
↪295,168

 max_pooling2d_19 (MaxPooling2D)    (None, 2, 2, 256)                   ␣
↪  0

 flatten_5 (Flatten)                (None, 1024)                        ␣
↪  0
```

```
dense_15 (Dense)                        (None, 128)                    ␣
↪131,200

dense_16 (Dense)                        (None, 64)                      ␣
↪8,256

dense_17 (Dense)                        (None, 10)                       ␣
↪650
```

Total params: 528,522 (2.02 MB)

Trainable params: 528,522 (2.02 MB)

Non-trainable params: 0 (0.00 B)

[21]:
```python
# Train both models
print("Training original model...")
history_original = original_model.fit(x_train, y_train, epochs=20,␣
 ↪batch_size=64, validation_data=(x_test, y_test))

print("\nTraining modified model with extra layer...")
history_modified = modified_model.fit(x_train, y_train, epochs=20,␣
 ↪batch_size=64, validation_data=(x_test, y_test))

# Final evaluation of both models
original_train_loss, original_train_accuracy = original_model.evaluate(x_train,␣
 ↪y_train, verbose=0)
original_test_loss, original_test_accuracy = original_model.evaluate(x_test,␣
 ↪y_test, verbose=0)

modified_train_loss, modified_train_accuracy = modified_model.evaluate(x_train,␣
 ↪y_train, verbose=0)
modified_test_loss, modified_test_accuracy = modified_model.evaluate(x_test,␣
 ↪y_test, verbose=0)

print(f"\nOriginal Model - Training Accuracy: {original_train_accuracy * 100:.
 ↪2f}%, Validation Accuracy: {original_test_accuracy * 100:.2f}%")
print(f"Modified Model - Training Accuracy: {modified_train_accuracy * 100:.
 ↪2f}%, Validation Accuracy: {modified_test_accuracy * 100:.2f}%")
```

```
Training original model…
Epoch 1/20
782/782              8s 7ms/step -
accuracy: 0.3675 - loss: 1.7192 - val_accuracy: 0.5833 - val_loss: 1.1669
```

```
Epoch 2/20
782/782            3s 4ms/step -
accuracy: 0.6146 - loss: 1.0810 - val_accuracy: 0.6660 - val_loss: 0.9457
Epoch 3/20
782/782            3s 4ms/step -
accuracy: 0.6929 - loss: 0.8788 - val_accuracy: 0.6971 - val_loss: 0.8663
Epoch 4/20
782/782            6s 5ms/step -
accuracy: 0.7366 - loss: 0.7561 - val_accuracy: 0.7114 - val_loss: 0.8464
Epoch 5/20
782/782            3s 4ms/step -
accuracy: 0.7653 - loss: 0.6681 - val_accuracy: 0.7060 - val_loss: 0.8413
Epoch 6/20
782/782            3s 4ms/step -
accuracy: 0.7929 - loss: 0.5953 - val_accuracy: 0.7247 - val_loss: 0.8239
Epoch 7/20
782/782            6s 4ms/step -
accuracy: 0.8160 - loss: 0.5192 - val_accuracy: 0.7338 - val_loss: 0.7769
Epoch 8/20
782/782            3s 4ms/step -
accuracy: 0.8372 - loss: 0.4650 - val_accuracy: 0.7081 - val_loss: 0.8968
Epoch 9/20
782/782            5s 4ms/step -
accuracy: 0.8542 - loss: 0.4124 - val_accuracy: 0.7441 - val_loss: 0.8293
Epoch 10/20
782/782            6s 5ms/step -
accuracy: 0.8745 - loss: 0.3531 - val_accuracy: 0.7388 - val_loss: 0.8725
Epoch 11/20
782/782            3s 4ms/step -
accuracy: 0.8887 - loss: 0.3138 - val_accuracy: 0.7350 - val_loss: 0.8881
Epoch 12/20
782/782            5s 4ms/step -
accuracy: 0.9053 - loss: 0.2687 - val_accuracy: 0.7264 - val_loss: 1.0087
Epoch 13/20
782/782            6s 5ms/step -
accuracy: 0.9137 - loss: 0.2409 - val_accuracy: 0.7304 - val_loss: 1.0467
Epoch 14/20
782/782            5s 4ms/step -
accuracy: 0.9251 - loss: 0.2102 - val_accuracy: 0.7312 - val_loss: 1.0711
Epoch 15/20
782/782            3s 4ms/step -
accuracy: 0.9372 - loss: 0.1771 - val_accuracy: 0.7285 - val_loss: 1.1660
Epoch 16/20
782/782            4s 5ms/step -
accuracy: 0.9371 - loss: 0.1743 - val_accuracy: 0.7397 - val_loss: 1.1625
Epoch 17/20
782/782            5s 4ms/step -
accuracy: 0.9479 - loss: 0.1455 - val_accuracy: 0.7301 - val_loss: 1.2991
```

```
Epoch 18/20
782/782            3s 4ms/step -
accuracy: 0.9557 - loss: 0.1266 - val_accuracy: 0.7257 - val_loss: 1.3095
Epoch 19/20
782/782            3s 4ms/step -
accuracy: 0.9558 - loss: 0.1253 - val_accuracy: 0.7281 - val_loss: 1.3956
Epoch 20/20
782/782            3s 4ms/step -
accuracy: 0.9540 - loss: 0.1290 - val_accuracy: 0.7231 - val_loss: 1.4876

Training modified model with extra layer…
Epoch 1/20
782/782            10s 9ms/step -
accuracy: 0.3286 - loss: 1.8056 - val_accuracy: 0.5919 - val_loss: 1.1287
Epoch 2/20
782/782            6s 5ms/step -
accuracy: 0.6081 - loss: 1.0840 - val_accuracy: 0.6652 - val_loss: 0.9514
Epoch 3/20
782/782            6s 5ms/step -
accuracy: 0.7041 - loss: 0.8353 - val_accuracy: 0.6772 - val_loss: 0.9418
Epoch 4/20
782/782            4s 5ms/step -
accuracy: 0.7581 - loss: 0.6899 - val_accuracy: 0.7352 - val_loss: 0.7802
Epoch 5/20
782/782            5s 5ms/step -
accuracy: 0.7986 - loss: 0.5678 - val_accuracy: 0.7387 - val_loss: 0.7886
Epoch 6/20
782/782            5s 5ms/step -
accuracy: 0.8321 - loss: 0.4727 - val_accuracy: 0.7475 - val_loss: 0.7569
Epoch 7/20
782/782            5s 5ms/step -
accuracy: 0.8653 - loss: 0.3860 - val_accuracy: 0.7434 - val_loss: 0.8130
Epoch 8/20
782/782            6s 6ms/step -
accuracy: 0.8911 - loss: 0.3119 - val_accuracy: 0.7436 - val_loss: 0.9003
Epoch 9/20
782/782            4s 5ms/step -
accuracy: 0.9121 - loss: 0.2498 - val_accuracy: 0.7461 - val_loss: 0.8984
Epoch 10/20
782/782            5s 5ms/step -
accuracy: 0.9313 - loss: 0.2011 - val_accuracy: 0.7321 - val_loss: 1.0650
Epoch 11/20
782/782            4s 5ms/step -
accuracy: 0.9363 - loss: 0.1824 - val_accuracy: 0.7425 - val_loss: 1.0817
Epoch 12/20
782/782            4s 5ms/step -
accuracy: 0.9512 - loss: 0.1411 - val_accuracy: 0.7470 - val_loss: 1.1716
Epoch 13/20
```

```
782/782              4s 5ms/step -
accuracy: 0.9534 - loss: 0.1338 - val_accuracy: 0.7447 - val_loss: 1.2081
Epoch 14/20
782/782              4s 6ms/step -
accuracy: 0.9618 - loss: 0.1075 - val_accuracy: 0.7412 - val_loss: 1.2350
Epoch 15/20
782/782              4s 5ms/step -
accuracy: 0.9642 - loss: 0.1027 - val_accuracy: 0.7350 - val_loss: 1.2969
Epoch 16/20
782/782              4s 5ms/step -
accuracy: 0.9653 - loss: 0.1012 - val_accuracy: 0.7422 - val_loss: 1.3481
Epoch 17/20
782/782              6s 5ms/step -
accuracy: 0.9675 - loss: 0.0923 - val_accuracy: 0.7394 - val_loss: 1.2960
Epoch 18/20
782/782              5s 5ms/step -
accuracy: 0.9704 - loss: 0.0847 - val_accuracy: 0.7387 - val_loss: 1.5074
Epoch 19/20
782/782              4s 5ms/step -
accuracy: 0.9709 - loss: 0.0835 - val_accuracy: 0.7412 - val_loss: 1.5056
Epoch 20/20
782/782              6s 5ms/step -
accuracy: 0.9740 - loss: 0.0790 - val_accuracy: 0.7336 - val_loss: 1.3954

Original Model - Training Accuracy: 94.59%, Validation Accuracy: 72.31%
Modified Model - Training Accuracy: 96.98%, Validation Accuracy: 73.36%
```

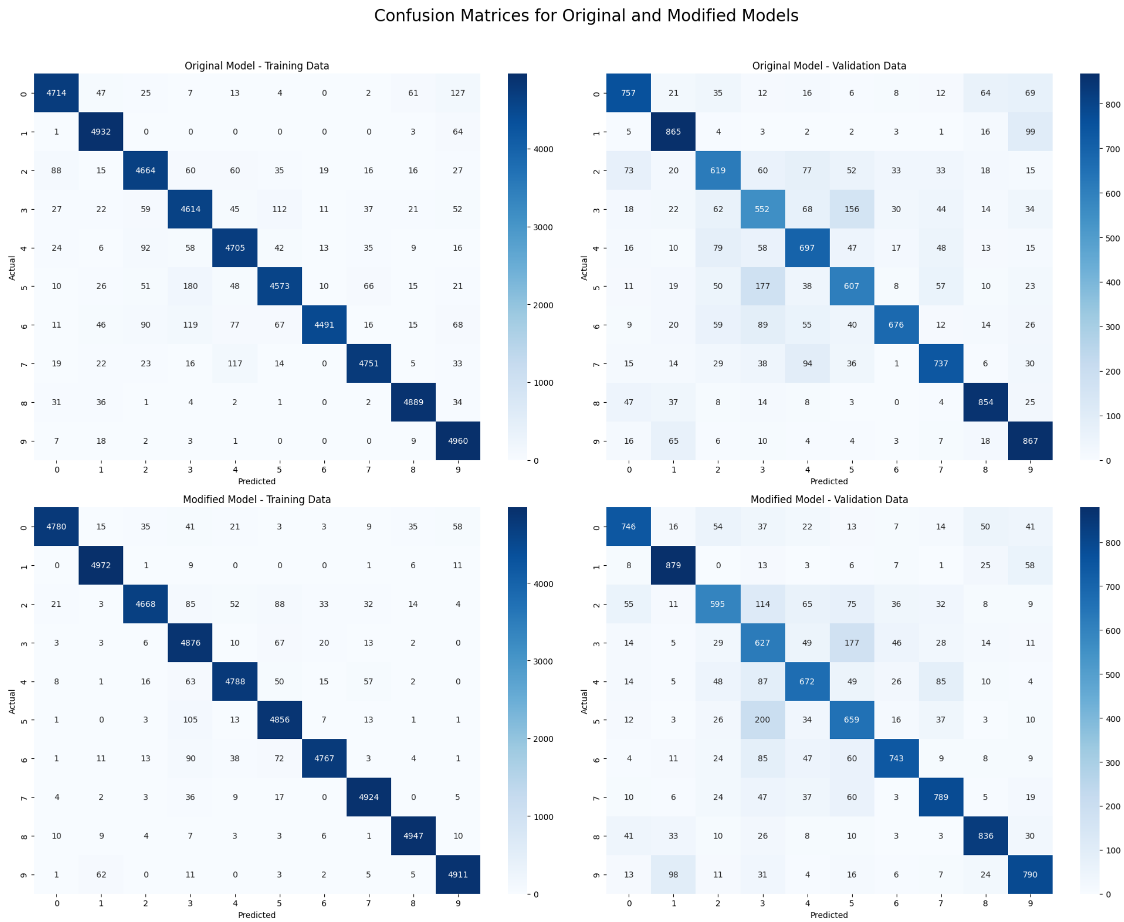
[22]:

```
def plot_confusion_matrices(model1, model2, x_train, y_train, x_test, y_test):
    fig, axes = plt.subplots(2, 2, figsize=(20, 16))
    fig.suptitle('Confusion Matrices for Original and Modified Models',␣
 ↪fontsize=20)

    # Original Model - Training Data
    y_pred_train_orig = model1.predict(x_train)
    y_pred_classes_train_orig = np.argmax(y_pred_train_orig, axis=1)
    y_true_train = np.argmax(y_train, axis=1)
    cm_train_orig = confusion_matrix(y_true_train, y_pred_classes_train_orig)

    sns.heatmap(cm_train_orig, annot=True, fmt='d', cmap='Blues', ax=axes[0,␣
 ↪0], xticklabels=range(10), yticklabels=range(10))
    axes[0, 0].set_title('Original Model - Training Data')
    axes[0, 0].set_xlabel('Predicted')
    axes[0, 0].set_ylabel('Actual')

    # Original Model - Validation Data
    y_pred_test_orig = model1.predict(x_test)
    y_pred_classes_test_orig = np.argmax(y_pred_test_orig, axis=1)
```

```
    y_true_test = np.argmax(y_test, axis=1)
    cm_test_orig = confusion_matrix(y_true_test, y_pred_classes_test_orig)

    sns.heatmap(cm_test_orig, annot=True, fmt='d', cmap='Blues', ax=axes[0, 1],␣
↪xticklabels=range(10), yticklabels=range(10))
    axes[0, 1].set_title('Original Model - Validation Data')
    axes[0, 1].set_xlabel('Predicted')
    axes[0, 1].set_ylabel('Actual')

    # Modified Model - Training Data
    y_pred_train_mod = model2.predict(x_train)
    y_pred_classes_train_mod = np.argmax(y_pred_train_mod, axis=1)
    cm_train_mod = confusion_matrix(y_true_train, y_pred_classes_train_mod)

    sns.heatmap(cm_train_mod, annot=True, fmt='d', cmap='Blues', ax=axes[1, 0],␣
↪xticklabels=range(10), yticklabels=range(10))
    axes[1, 0].set_title('Modified Model - Training Data')
    axes[1, 0].set_xlabel('Predicted')
    axes[1, 0].set_ylabel('Actual')

    # Modified Model - Validation Data
    y_pred_test_mod = model2.predict(x_test)
    y_pred_classes_test_mod = np.argmax(y_pred_test_mod, axis=1)
    cm_test_mod = confusion_matrix(y_true_test, y_pred_classes_test_mod)

    sns.heatmap(cm_test_mod, annot=True, fmt='d', cmap='Blues', ax=axes[1, 1],␣
↪xticklabels=range(10), yticklabels=range(10))
    axes[1, 1].set_title('Modified Model - Validation Data')
    axes[1, 1].set_xlabel('Predicted')
    axes[1, 1].set_ylabel('Actual')

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()
```

[23]:
```
plot_confusion_matrices(original_model, modified_model, x_train, y_train,␣
↪x_test, y_test)
```

```
1563/1563           3s 2ms/step
313/313           1s 2ms/step
1563/1563           3s 2ms/step
313/313           1s 2ms/step
```

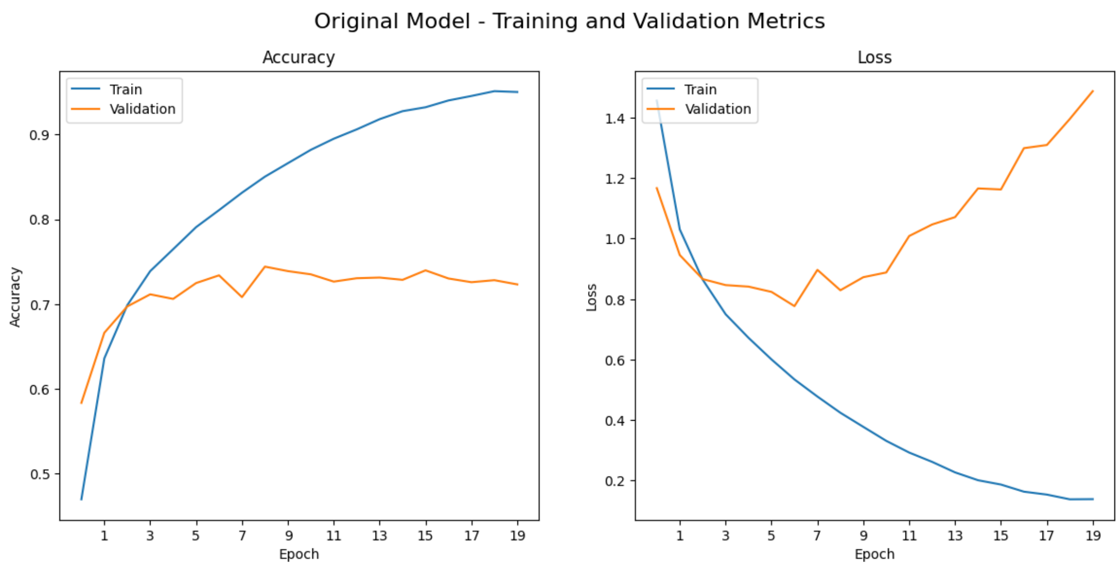


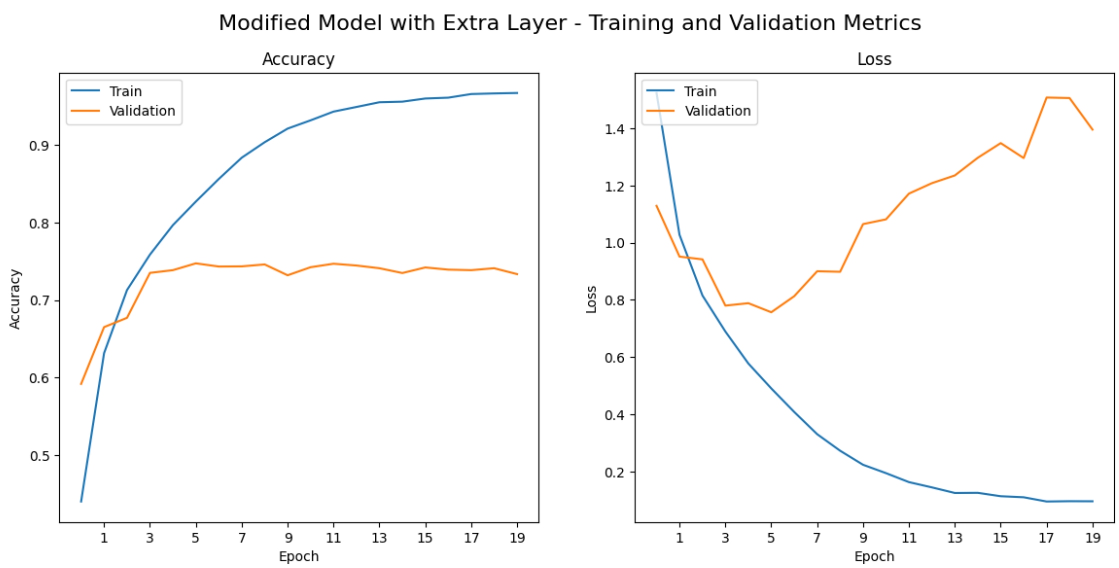

```
[24]: def plot_training_history(history, title="Model"):
          # Plot training & validation accuracy values
          plt.figure(figsize=(14, 6))
          plt.suptitle(f'{title} - Training and Validation Metrics', fontsize=16)

          plt.subplot(1, 2, 1)
          plt.plot(history.history['accuracy'])
          plt.plot(history.history['val_accuracy'])
          plt.title('Accuracy')
          plt.ylabel('Accuracy')
          plt.xlabel('Epoch')
          plt.xticks(np.arange(1, len(history.history['accuracy'])+1, 2))
          plt.legend(['Train', 'Validation'], loc='upper left')

          # Plot training & validation loss values
          plt.subplot(1, 2, 2)
          plt.plot(history.history['loss'])
          plt.plot(history.history['val_loss'])
```

```
    plt.title('Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.xticks(np.arange(1, len(history.history['accuracy'])+1, 2))
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.show()
```

[25]:

```
# Plot training history for both models
plot_training_history(history_original, title="Original Model")
plot_training_history(history_modified, title="Modified Model with Extra Layer")
```





2 Problem 2

[26]:
```python
# Load and preprocess CIFAR-10 data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize the pixel values to be between 0 and 1
x_train, x_test = x_train.astype('float32') / 255.0, x_test.astype('float32') /
↪255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Define the original CNN model without batch normalization
def create_original_cnn_model():
    model = models.Sequential()

    # First convolutional block
    model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same',
↪input_shape=(32, 32, 3)))
    model.add(layers.MaxPooling2D((2, 2)))

    # Second convolutional block
    model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Third convolutional block
    model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Flatten and add dense layers
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(64, activation='relu'))

    # Output layer
    model.add(layers.Dense(10, activation='softmax'))  # 10 classes for CIFAR-10

    return model

# Define the original CNN model with batch normalization
def create_cnn_model_with_batch_norm():
    model = models.Sequential()

    # First convolutional block
```

```python
    model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same',␣
↪input_shape=(32, 32, 3)))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D((2, 2)))

    # Second convolutional block
    model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D((2, 2)))

    # Third convolutional block
    model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D((2, 2)))

    # Flatten and add dense layers
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.BatchNormalization())

    # Output layer
    model.add(layers.Dense(10, activation='softmax'))  # 10 classes for CIFAR-10

    return model
```

[27]:
```python
# Instantiate both models
original_model = create_original_cnn_model()
original_model_bn = create_cnn_model_with_batch_norm()

# Compile both models
original_model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
original_model_bn.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[28]:
```python
# Train both models and record training time
import time
```

```python
print("Training original model...")
start_time = time.time()
history_original = original_model.fit(x_train, y_train, epochs=20,␣
↪batch_size=64, validation_data=(x_test, y_test))
end_time = time.time()
original_training_time = end_time - start_time
print(f"Original model training completed in {original_training_time:.2f}␣
↪seconds.")

print("\nTraining original model with batch normalization...")
start_time = time.time()
history_original_bn = original_model_bn.fit(x_train, y_train, epochs=20,␣
↪batch_size=64, validation_data=(x_test, y_test))
end_time = time.time()
original_bn_training_time = end_time - start_time
print(f"Original model with batch normalization training completed in␣
↪{original_bn_training_time:.2f} seconds.")
```

```
Training original model…
Epoch 1/20
782/782              7s 6ms/step -
accuracy: 0.3347 - loss: 1.7947 - val_accuracy: 0.5533 - val_loss: 1.2140
Epoch 2/20
782/782              8s 4ms/step -
accuracy: 0.5932 - loss: 1.1288 - val_accuracy: 0.6531 - val_loss: 0.9881
Epoch 3/20
782/782              5s 4ms/step -
accuracy: 0.6836 - loss: 0.8978 - val_accuracy: 0.6830 - val_loss: 0.9038
Epoch 4/20
782/782              4s 5ms/step -
accuracy: 0.7292 - loss: 0.7706 - val_accuracy: 0.7093 - val_loss: 0.8360
Epoch 5/20
782/782              3s 4ms/step -
accuracy: 0.7592 - loss: 0.6844 - val_accuracy: 0.7167 - val_loss: 0.8283
Epoch 6/20
782/782              3s 4ms/step -
accuracy: 0.7927 - loss: 0.5953 - val_accuracy: 0.7377 - val_loss: 0.7625
Epoch 7/20
782/782              6s 4ms/step -
accuracy: 0.8137 - loss: 0.5381 - val_accuracy: 0.7426 - val_loss: 0.7707
Epoch 8/20
782/782              5s 4ms/step -
accuracy: 0.8365 - loss: 0.4649 - val_accuracy: 0.7516 - val_loss: 0.7567
Epoch 9/20
782/782              3s 4ms/step -
accuracy: 0.8594 - loss: 0.4066 - val_accuracy: 0.7440 - val_loss: 0.8107
Epoch 10/20
```

```
782/782            3s 4ms/step -
accuracy: 0.8726 - loss: 0.3576 - val_accuracy: 0.7460 - val_loss: 0.8099
Epoch 11/20
782/782            4s 5ms/step -
accuracy: 0.8888 - loss: 0.3185 - val_accuracy: 0.7395 - val_loss: 0.9044
Epoch 12/20
782/782            5s 4ms/step -
accuracy: 0.9049 - loss: 0.2757 - val_accuracy: 0.7388 - val_loss: 0.9443
Epoch 13/20
782/782            5s 4ms/step -
accuracy: 0.9150 - loss: 0.2426 - val_accuracy: 0.7483 - val_loss: 0.9468
Epoch 14/20
782/782            4s 4ms/step -
accuracy: 0.9283 - loss: 0.2037 - val_accuracy: 0.7400 - val_loss: 0.9998
Epoch 15/20
782/782            3s 4ms/step -
accuracy: 0.9362 - loss: 0.1821 - val_accuracy: 0.7358 - val_loss: 1.1077
Epoch 16/20
782/782            3s 4ms/step -
accuracy: 0.9418 - loss: 0.1684 - val_accuracy: 0.7370 - val_loss: 1.2159
Epoch 17/20
782/782            5s 6ms/step -
accuracy: 0.9500 - loss: 0.1451 - val_accuracy: 0.7367 - val_loss: 1.2360
Epoch 18/20
782/782            8s 4ms/step -
accuracy: 0.9484 - loss: 0.1448 - val_accuracy: 0.7335 - val_loss: 1.3886
Epoch 19/20
782/782            3s 4ms/step -
accuracy: 0.9530 - loss: 0.1311 - val_accuracy: 0.7265 - val_loss: 1.3616
Epoch 20/20
782/782            5s 4ms/step -
accuracy: 0.9562 - loss: 0.1264 - val_accuracy: 0.7320 - val_loss: 1.3932
Original model training completed in 92.96 seconds.

Training original model with batch normalization…
Epoch 1/20
782/782            13s 9ms/step -
accuracy: 0.4834 - loss: 1.4602 - val_accuracy: 0.4436 - val_loss: 1.7126
Epoch 2/20
782/782            4s 5ms/step -
accuracy: 0.6974 - loss: 0.8734 - val_accuracy: 0.6940 - val_loss: 0.8784
Epoch 3/20
782/782            6s 6ms/step -
accuracy: 0.7659 - loss: 0.6795 - val_accuracy: 0.6957 - val_loss: 0.8805
Epoch 4/20
782/782            4s 5ms/step -
accuracy: 0.8128 - loss: 0.5520 - val_accuracy: 0.7225 - val_loss: 0.8184
Epoch 5/20
```

```
782/782              6s 5ms/step -
accuracy: 0.8470 - loss: 0.4398 - val_accuracy: 0.7223 - val_loss: 0.8515
Epoch 6/20
782/782              4s 5ms/step -
accuracy: 0.8808 - loss: 0.3489 - val_accuracy: 0.7451 - val_loss: 0.8040
Epoch 7/20
782/782              5s 5ms/step -
accuracy: 0.8993 - loss: 0.2917 - val_accuracy: 0.7352 - val_loss: 0.9167
Epoch 8/20
782/782              4s 5ms/step -
accuracy: 0.9260 - loss: 0.2203 - val_accuracy: 0.6903 - val_loss: 1.1059
Epoch 9/20
782/782              5s 5ms/step -
accuracy: 0.9378 - loss: 0.1810 - val_accuracy: 0.7003 - val_loss: 1.1159
Epoch 10/20
782/782              5s 5ms/step -
accuracy: 0.9443 - loss: 0.1606 - val_accuracy: 0.7254 - val_loss: 1.0847
Epoch 11/20
782/782              6s 5ms/step -
accuracy: 0.9523 - loss: 0.1402 - val_accuracy: 0.7341 - val_loss: 1.0977
Epoch 12/20
782/782              4s 5ms/step -
accuracy: 0.9565 - loss: 0.1224 - val_accuracy: 0.6926 - val_loss: 1.3914
Epoch 13/20
782/782              5s 5ms/step -
accuracy: 0.9581 - loss: 0.1191 - val_accuracy: 0.7357 - val_loss: 1.1727
Epoch 14/20
782/782              5s 5ms/step -
accuracy: 0.9652 - loss: 0.0982 - val_accuracy: 0.7436 - val_loss: 1.1696
Epoch 15/20
782/782              5s 5ms/step -
accuracy: 0.9716 - loss: 0.0845 - val_accuracy: 0.7512 - val_loss: 1.1373
Epoch 16/20
782/782              4s 6ms/step -
accuracy: 0.9694 - loss: 0.0876 - val_accuracy: 0.7448 - val_loss: 1.1655
Epoch 17/20
782/782              4s 5ms/step -
accuracy: 0.9707 - loss: 0.0846 - val_accuracy: 0.6923 - val_loss: 1.6226
Epoch 18/20
782/782              5s 5ms/step -
accuracy: 0.9712 - loss: 0.0830 - val_accuracy: 0.7434 - val_loss: 1.2910
Epoch 19/20
782/782              5s 5ms/step -
accuracy: 0.9752 - loss: 0.0694 - val_accuracy: 0.7393 - val_loss: 1.3204
Epoch 20/20
782/782              4s 5ms/step -
accuracy: 0.9756 - loss: 0.0712 - val_accuracy: 0.7419 - val_loss: 1.2885
Original model with batch normalization training completed in 105.26 seconds.
```

[29]:

```
# Final evaluation of both models
original_train_loss, original_train_accuracy = original_model.evaluate(x_train,␣
↪y_train, verbose=0)
original_test_loss, original_test_accuracy = original_model.evaluate(x_test,␣
↪y_test, verbose=0)

original_bn_train_loss, original_bn_train_accuracy = original_model_bn.
↪evaluate(x_train, y_train, verbose=0)
original_bn_test_loss, original_bn_test_accuracy = original_model_bn.
↪evaluate(x_test, y_test, verbose=0)

# Print training and validation accuracy
print(f"\nOriginal Model - Training Accuracy: {original_train_accuracy:.4f},␣
↪Validation Accuracy: {original_test_accuracy:.4f}")
print(f"Original Model with Batch Normalization - Training Accuracy:␣
↪{original_bn_train_accuracy:.4f}, Validation Accuracy:␣
↪{original_bn_test_accuracy:.4f}")

# Print training times
print(f"Original Model Training Time: {original_training_time:.2f} seconds")
print(f"Original Model with Batch Normalization Training Time:␣
↪{original_bn_training_time:.2f} seconds")
```

```
Original Model - Training Accuracy: 0.9639, Validation Accuracy: 0.7320
Original Model with Batch Normalization - Training Accuracy: 0.9749, Validation
Accuracy: 0.7419
Original Model Training Time: 92.96 seconds
Original Model with Batch Normalization Training Time: 105.26 seconds
```

[30]:

```
# Plot training and validation accuracy for both models on the same plot
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Plot training accuracy
plt.plot(history_original.history['accuracy'], label='Original Model - Training␣
↪Accuracy', linestyle='--')
plt.plot(history_original_bn.history['accuracy'], label='Original Model with BN␣
↪- Training Accuracy', linestyle='--')

# Plot validation accuracy
plt.plot(history_original.history['val_accuracy'], label='Original Model -␣
↪Validation Accuracy')
plt.plot(history_original_bn.history['val_accuracy'], label='Original Model␣
↪with BN - Validation Accuracy')
```

```
plt.title('Training and Validation Accuracy for Original Model and Model with␣
 ↪Batch Normalization')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```