SYDE_522_A4_Part_2_CIFAR_10

November 29, 2023

0.0.1 Introduction

[1]:
```
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',␣
 ↪'horse', 'ship', 'truck']

# Scale pixel values to be between 0 and 1
x_train = x_train / 255.0
x_test = x_test / 255.0
y_test = y_test[:,0]
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

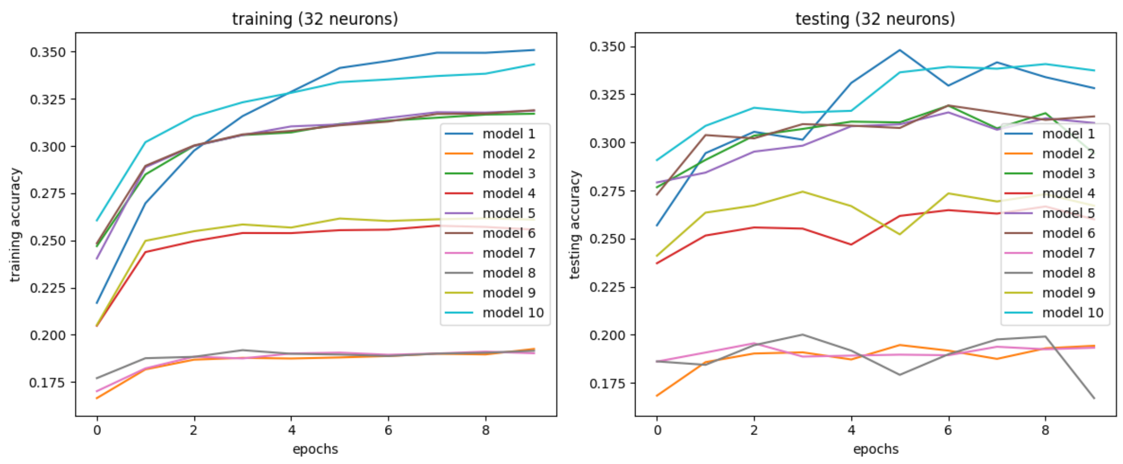
[2]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(14,6))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_train[i])
    plt.xticks([])
    plt.yticks([])
    plt.title(names[int(y_train[i])])
plt.show()
```

0.0.2 Question 9

```
[ ]: models = {} # empty dict for storing 10 models
     for i in range(10):
       # instantiate new model
       model = tf.keras.models.Sequential([
         tf.keras.layers.Flatten(input_shape=(32, 32, 3)),    # input is a 32x32x3␣
       ↪image
         tf.keras.layers.Dense(32, activation='relu'),        # 32 neurons in the␣
       ↪middle "hidden" layer
         tf.keras.layers.Dense(10, activation='softmax')      # 10 outputs (one for␣
       ↪each category)
       ])
       model.compile(optimizer='adam',
                     loss='sparse_categorical_crossentropy',
                     metrics=['accuracy']  # in addition to the loss, also compute␣
       ↪the categorization accuracy
                     )
       # add to dict
       models[str(i)] = model

     # iterate through dict to train for 10 epochs
     training_acc = []
     testing_acc = []
     for key, model in models.items():
       print('\n')
       print(f'Model {key}:')
       model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))    #␣
       ↪note that we now use y_train, not y_train_target
       # append training and testing accuracy vs. epoch lists for each model
```

```
  training_acc.append(model.history.history['accuracy'])
  testing_acc.append(model.history.history['val_accuracy'])
```

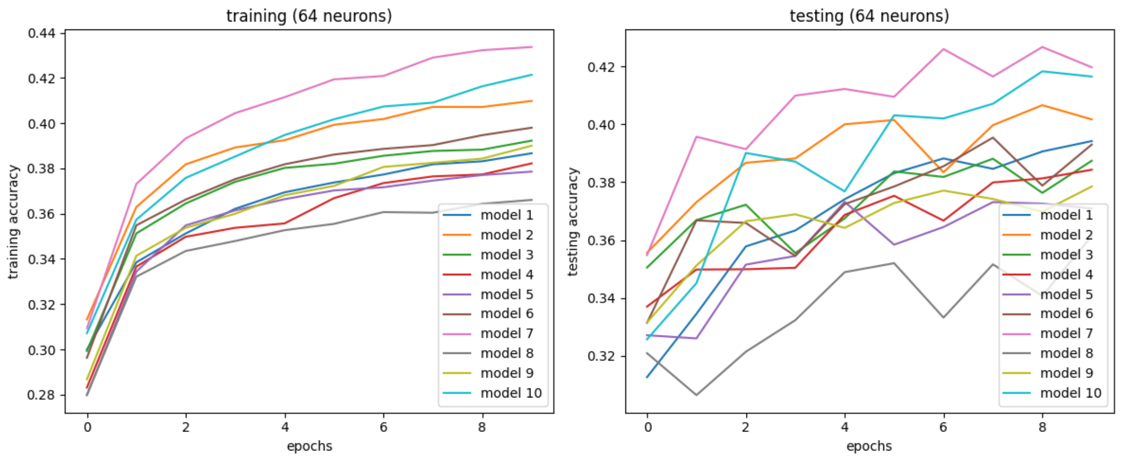
[ ]:
```
# plot each model's training accuracy
plt.figure(figsize=(12,5))
plt.subplot(1, 2, 1)
for model_idx, loss in enumerate(training_acc):
    plt.plot(loss, label=f'model {model_idx + 1}')

# add labels and legend
plt.xlabel('epochs')
plt.ylabel('training accuracy')
plt.title('training (32 neurons)')
plt.legend()

# plot each model's testing accuracy
plt.subplot(1, 2, 2)
for model_idx, loss in enumerate(testing_acc):
    plt.plot(loss, label=f'model {model_idx + 1}')

# add labels and legend
plt.xlabel('epochs')
plt.ylabel('testing accuracy')
plt.title('testing (32 neurons)')
plt.legend()

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



[ ]:
```
models = {} # empty dict for storing 10 models
for i in range(10):
  # instantiate new model
  model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(32, 32, 3)),   # input is a 32x32x3␣
 ↪image
    tf.keras.layers.Dense(64, activation='relu'),       # 64 neurons in the␣
 ↪middle "hidden" layer
    tf.keras.layers.Dense(10, activation='softmax')     # 10 outputs (one for␣
 ↪each category)
  ])
  model.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy']  # in addition to the loss, also compute␣
 ↪the categorization accuracy
                )
  # add to dict
  models[str(i)] = model

# iterate through dict to train for 10 epochs
training_acc = []
testing_acc = []
for key, model in models.items():
  print('\n')
  print(f'Model {key}:')
  model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))   #␣
 ↪note that we now use y_train, not y_train_target
  # append training and testing accuracy vs. epoch lists for each model
  training_acc.append(model.history.history['accuracy'])
  testing_acc.append(model.history.history['val_accuracy'])
```

[ ]:
```
# plot each model's training accuracy
plt.figure(figsize=(12,5))
plt.subplot(1, 2, 1)
for model_idx, loss in enumerate(training_acc):
    plt.plot(loss, label=f'model {model_idx + 1}')

# add labels and legend
plt.xlabel('epochs')
plt.ylabel('training accuracy')
plt.title('training (64 neurons)')
plt.legend()

# plot each model's testing accuracy
plt.subplot(1, 2, 2)
for model_idx, loss in enumerate(testing_acc):
    plt.plot(loss, label=f'model {model_idx + 1}')

# add labels and legend
```

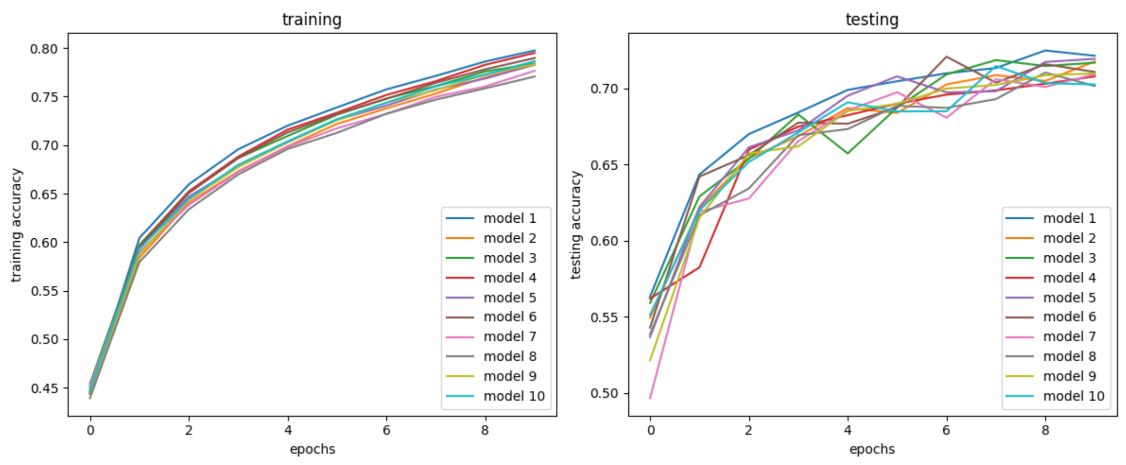
```python
plt.xlabel('epochs')
plt.ylabel('testing accuracy')
plt.title('testing (64 neurons)')
plt.legend()

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



0.0.3 Question 9

[ ]:
```python
models = {} # empty dict for storing 10 models
for i in range(10):
  # instantiate new model
  model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,␣
↪3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
  ])
  model.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy']  # in addition to the loss, also compute␣
↪the categorization accuracy
                )
  # add to dict
  models[str(i)] = model

# iterate through dict to train for 10 epochs
training_acc = []
testing_acc = []
for key, model in models.items():
  print('\n')
  print(f'Model {key}:')
  model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))   #␣
↪note that we now use y_train, not y_train_target
  # append training and testing accuracy vs. epoch lists for each model
  training_acc.append(model.history.history['accuracy'])
  testing_acc.append(model.history.history['val_accuracy'])
```

```
[ ]: # plot each model's training accuracy
plt.figure(figsize=(12,5))
plt.subplot(1, 2, 1)
for model_idx, loss in enumerate(training_acc):
    plt.plot(loss, label=f'model {model_idx + 1}')

# add labels and legend
plt.xlabel('epochs')
plt.ylabel('training accuracy')
plt.title('training')
plt.legend()

# plot each model's testing accuracy
plt.subplot(1, 2, 2)
for model_idx, loss in enumerate(testing_acc):
    plt.plot(loss, label=f'model {model_idx + 1}')

# add labels and legend
plt.xlabel('epochs')
plt.ylabel('testing accuracy')
plt.title('testing')
plt.legend()

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

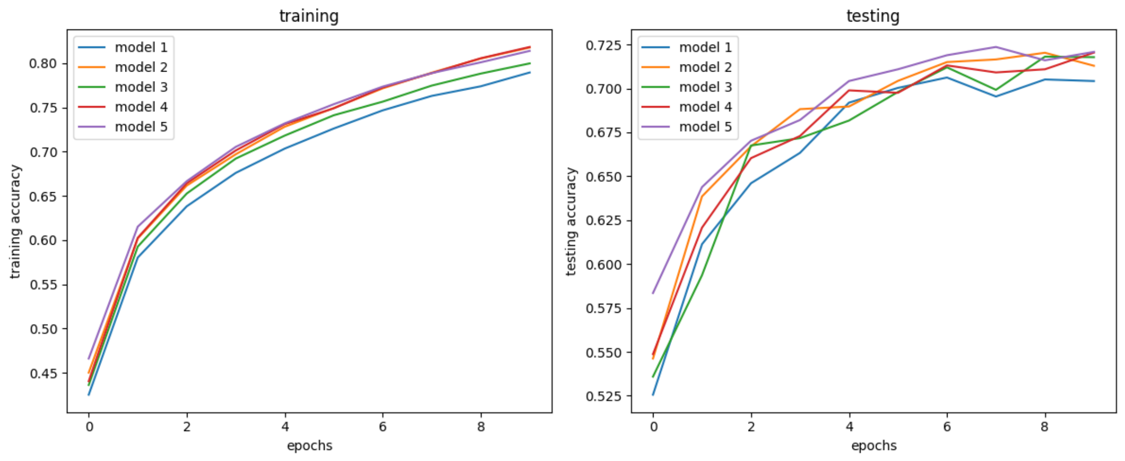


0.0.4 Question 10

Exploring an additional Conv2D and MaxPooling2D layer

```
[ ]: models = {}  # empty dict for storing 5 models
for i in range(5):
    # instantiate new model
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32,␣
 ↪32, 3)),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),  # additional␣
 ↪Conv2D layer
        tf.keras.layers.MaxPooling2D((2, 2)),                    # additional␣
 ↪MaxPooling2D layer
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    # add to dict
    models[str(i)] = model

# iterate through dict to train for 10 epochs
training_acc = []
testing_acc = []
for key, model in models.items():
    print('\n')
    print(f'Model {key}:')
    model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))
    # append training and testing accuracy vs. epoch lists for each model
    training_acc.append(model.history.history['accuracy'])
    testing_acc.append(model.history.history['val_accuracy'])
```

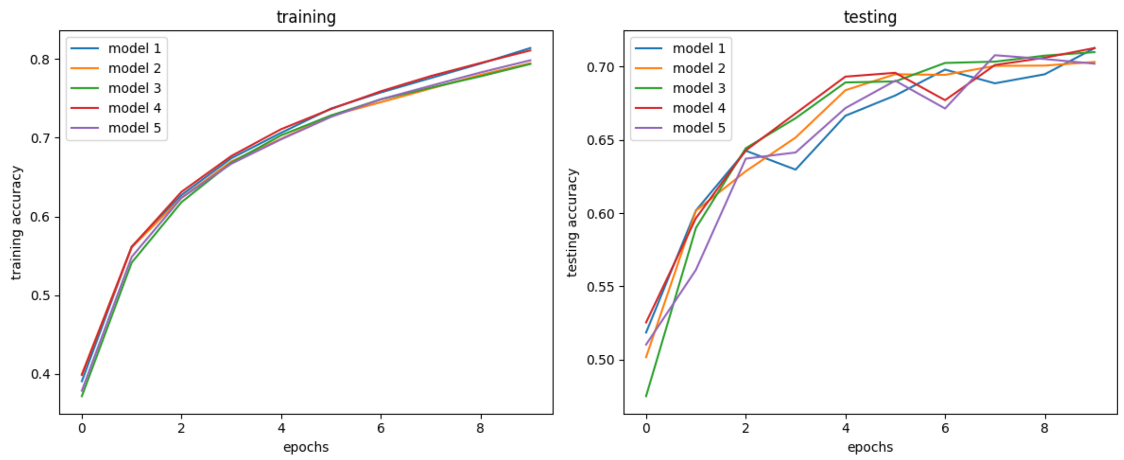
[4]:
```
# plot each model's training accuracy
plt.figure(figsize=(12,5))
plt.subplot(1, 2, 1)
for model_idx, loss in enumerate(training_acc):
    plt.plot(loss, label=f'model {model_idx + 1}')

# add labels and legend
plt.xlabel('epochs')
plt.ylabel('training accuracy')
plt.title('training')
plt.legend()

# plot each model's testing accuracy
plt.subplot(1, 2, 2)
for model_idx, loss in enumerate(testing_acc):
    plt.plot(loss, label=f'model {model_idx + 1}')

# add labels and legend
plt.xlabel('epochs')
plt.ylabel('testing accuracy')
plt.title('testing')
plt.legend()

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



Exploring two additional Conv2D and MaxPooling2D layers

[ ]:
```
models = {}  # empty dict for storing 5 models
for i in range(5):
    # instantiate new model
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32,␣
 ↪32, 3)),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),  # additional␣
 ↪Conv2D layer
        tf.keras.layers.MaxPooling2D((2, 2)),                    # additional␣
 ↪MaxPooling2D layer
        tf.keras.layers.Conv2D(256, (1, 1), activation='relu'),  # additional␣
 ↪Conv2D layer
        tf.keras.layers.MaxPooling2D((2, 2)),                    # additional␣
 ↪MaxPooling2D layer
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    # add to dict
    models[str(i)] = model

# iterate through dict to train for 10 epochs
training_acc = []
testing_acc = []
for key, model in models.items():
    print('\n')
    print(f'Model {key}:')
    model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))
    # append training and testing accuracy vs. epoch lists for each model
    training_acc.append(model.history.history['accuracy'])
    testing_acc.append(model.history.history['val_accuracy'])
```

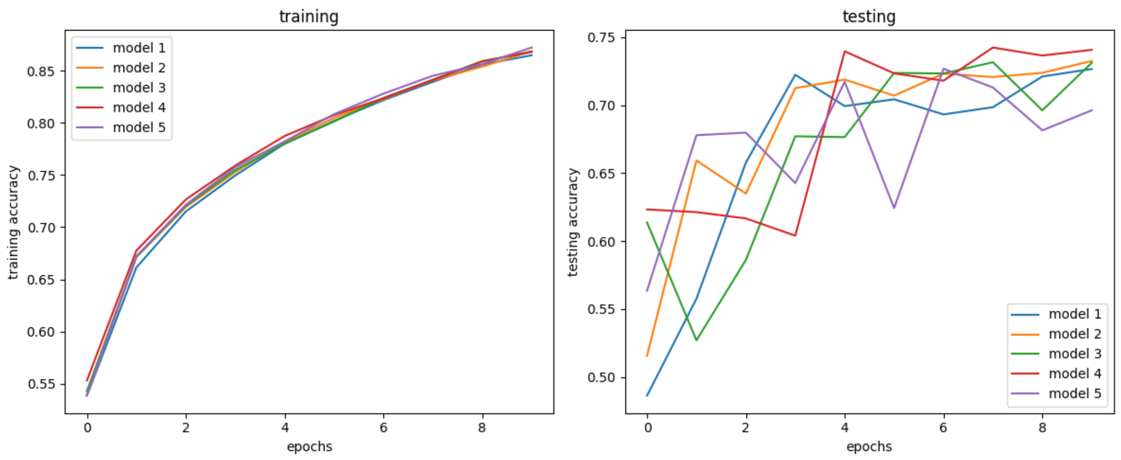
[11]:
```
# plot each model's training accuracy
plt.figure(figsize=(12,5))
plt.subplot(1, 2, 1)
for model_idx, loss in enumerate(training_acc):
    plt.plot(loss, label=f'model {model_idx + 1}')

# add labels and legend
plt.xlabel('epochs')
plt.ylabel('training accuracy')
plt.title('training')
plt.legend()

# plot each model's testing accuracy
plt.subplot(1, 2, 2)
for model_idx, loss in enumerate(testing_acc):
    plt.plot(loss, label=f'model {model_idx + 1}')

# add labels and legend
plt.xlabel('epochs')
plt.ylabel('testing accuracy')
plt.title('testing')
plt.legend()

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



Exploring batch normalization with one additional Conv2D and MaxPooling2D layer

[ ]:
```
models = {}  # empty dict for storing 5 models
for i in range(5):
    # instantiate new model
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32,␣
 ↪32, 3)),
        tf.keras.layers.BatchNormalization(),  # batch normalization
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
        tf.keras.layers.BatchNormalization(),  # batch normalization
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
        tf.keras.layers.BatchNormalization(),  # batch normalization
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.BatchNormalization(),  # batch normalization
        tf.keras.layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    # add to dict
    models[str(i)] = model

# iterate through dict to train for 10 epochs
training_acc = []
testing_acc = []
for key, model in models.items():
    print('\n')
    print(f'Model {key}:')
    model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))
    # append training and testing accuracy vs. epoch lists for each model
    training_acc.append(model.history.history['accuracy'])
    testing_acc.append(model.history.history['val_accuracy'])
```

[7]:
```
# plot each model's training accuracy
plt.figure(figsize=(12,5))
plt.subplot(1, 2, 1)
for model_idx, loss in enumerate(training_acc):
    plt.plot(loss, label=f'model {model_idx + 1}')

# add labels and legend
plt.xlabel('epochs')
plt.ylabel('training accuracy')
plt.title('training')
plt.legend()

# plot each model's testing accuracy
plt.subplot(1, 2, 2)
for model_idx, loss in enumerate(testing_acc):
    plt.plot(loss, label=f'model {model_idx + 1}')

# add labels and legend
plt.xlabel('epochs')
plt.ylabel('testing accuracy')
plt.title('testing')
plt.legend()
plt.tight_layout()
plt.show()
```



Analyzing results: To improve the model, I experimented with adding complexity with the addition of Conv2D and MaxPooling2D layers. While the testing accuracies do not differ significantly, the convergence of solutions improves with the addition of layers. This suggests that the enhanced model complexity enables more effective learning during training. I imagine that with some additional tuning, the testing accuracy could significantly improve as well. I also explored the use of batch normalization, which should stabilize and accelerate training by normalizing the inputs to each layer. Surprisingly, the performance on testing data worsened, indicating that the introduced normalization may not be beneficial for this specific architecture or dataset. Further investigation and fine-tuning would be useful in understanding the interplay between batch normalization and the model's performance.