Spilios Spiliopoulos 4495

Set 1 Machine Learning CSE

Step 1: Import Libraries and Load Fashion MNIST Data

- Load data from Fashion MNIST dataset
- Normalize data to range (0-1) instead of range (0-255)
- From Keras documentation at @: https://keras.io/api/datasets/fashion_mnist/

In [17]:
```
import numpy as np
from keras.datasets import fashion_mnist

# Load Fashion MNIST data
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data(
assert train_images.shape == (60000, 28, 28)
assert test_images.shape == (10000, 28, 28)
assert train_labels.shape == (60000,)
assert test_labels.shape == (10000,)

# Normalize the data
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0
```

Step 2: Downsample Images using Max Pooling

- Max Pooling with block_reduce function with stride = 4 in order to achieve (7,7)
- For each image of the training/testing dataset
- -> downsample to (7,7) for faster computation time and less parameters
- (the max value of each block is the value that will represent the same block)

In [18]:
```
from skimage.measure import block_reduce

# Reducing the dimension of images using max pooling 4x4
train_images_downsampled = np.zeros((train_images.shape[0], 7, 7))
test_images_downsampled = np.zeros((test_images.shape[0], 7, 7))

# For each image of the training dataset
for i in range(train_images.shape[0]):
    train_images_downsampled[i] = block_reduce(train_images[i], (4, 4), np.max)
```

```python
# For each image of the testing dataset
for i in range(test_images.shape[0]):
    test_images_downsampled[i] = block_reduce(test_images[i], (4, 4), np.max)
```

Sampling Training Data

- In order to achieve less computation time, we modify the samples from 60000 to 30000 (50%)
- (We can modify the percentage to keep = 1 in order to train the model with 60000 samples)
- Define percentage of the dataset
- For each category:
- Find all the indeces of data that belong to that category
- Multiply with percentage in order to keep the appropriate sample
- Select random indices == to the number of our sample dataset (50% of indices)
- Store the samples of those indeces to the appropriate lists:
- a) list for image values
- b) list for indeces of those values that are equal to the label of the images in list a)
- Then combine results of those arrays to one array respectively
- Shuffle the data in order to insert randomness and prevent bias

In [19]:
```python
# Import necessary libraries for sampling data
import numpy as np

# Define the percentage of data to keep (50%)
percentage_to_keep = 0.5

# Initialize lists to store the sampled data
sampled_train_images = []
sampled_train_labels = []

# Iterate over each category
for category in range(10):  # There are 10 categories in Fashion MNIST
    # Find indices of data belonging to the current category
    category_indices = np.where(train_labels == category)[0]

    # Randomly sample a subset of data from the current category
    num_samples = int(len(category_indices) * percentage_to_keep)
    sampled_indices = np.random.choice(category_indices, num_samples, replace=False

    # Add the sampled data to the lists
    sampled_train_images.append(train_images_downsampled[sampled_indices])
    sampled_train_labels.append(train_labels[sampled_indices])

# Concatenate the sampled data from all categories
sampled_train_images = np.concatenate(sampled_train_images, axis=0)
sampled_train_labels = np.concatenate(sampled_train_labels, axis=0)
```

```
# Shuffle the sampled data
shuffle_indices = np.random.permutation(len(sampled_train_images))
sampled_train_images = sampled_train_images[shuffle_indices]
sampled_train_labels = sampled_train_labels[shuffle_indices]

# Print the size of the sampled training set
print("Size of the sampled training set:", sampled_train_images.shape)
```

```
Size of the sampled training set: (30000, 7, 7)
```

Step 3: Vectorize Images

- Reshape the sample_train_images/test_image from 7x7 (2D array) to 1 x 49 (1D array) format
- This vector format is compatible for the machine learning algorithms.

In [20]:

```
print("Size of the sampled training set before dimension reduction:", sampled_train
print("Size of the testing set before dimension reduction:", test_images.shape)

# Conversion of sampled images into vector format
sampled_train_images_vectorized = sampled_train_images.reshape(sampled_train_images
test_images_vectorized = test_images_downsampled.reshape(test_images_downsampled.sh

print("Final size of the sampled training set after dimension reduction:", sampled_
print("Final size of the testing set after dimension reduction:", test_images_vecto
```

```
Size of the sampled training set before dimension reduction: (30000, 7, 7)
Size of the testing set before dimension reduction: (10000, 28, 28)
Final size of the sampled training set after dimension reduction: (30000, 49)
Final size of the testing set after dimension reduction: (10000, 49)
```

Step 4: K-Means Classification

Implementation of the KMeans algorithm:

- For k = 1,3,5 neighbors
- With Euclidean distance

In [21]:

```
################################ K MEANS ##########################################
from sklearn.neighbors import KNeighborsClassifier

# Create the k-NN classifier with Euclidean distance
k_values = [1, 3, 5]
for k in k_values:
    knn_classifier = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
    knn_classifier.fit(sampled_train_images_vectorized, sampled_train_labels)

    # Evaluate the model on the testing set
    accuracy = knn_classifier.score(test_images_vectorized, test_labels)
```

```
    print("Accuracy for k =", k, ":", accuracy)
#####################################################################################
```

```
Accuracy for k = 1 : 0.7563
Accuracy for k = 3 : 0.7699
Accuracy for k = 5 : 0.7749
```

Step 5: Decision Tree and Random Forest

Step 5.1 : Decision Tree

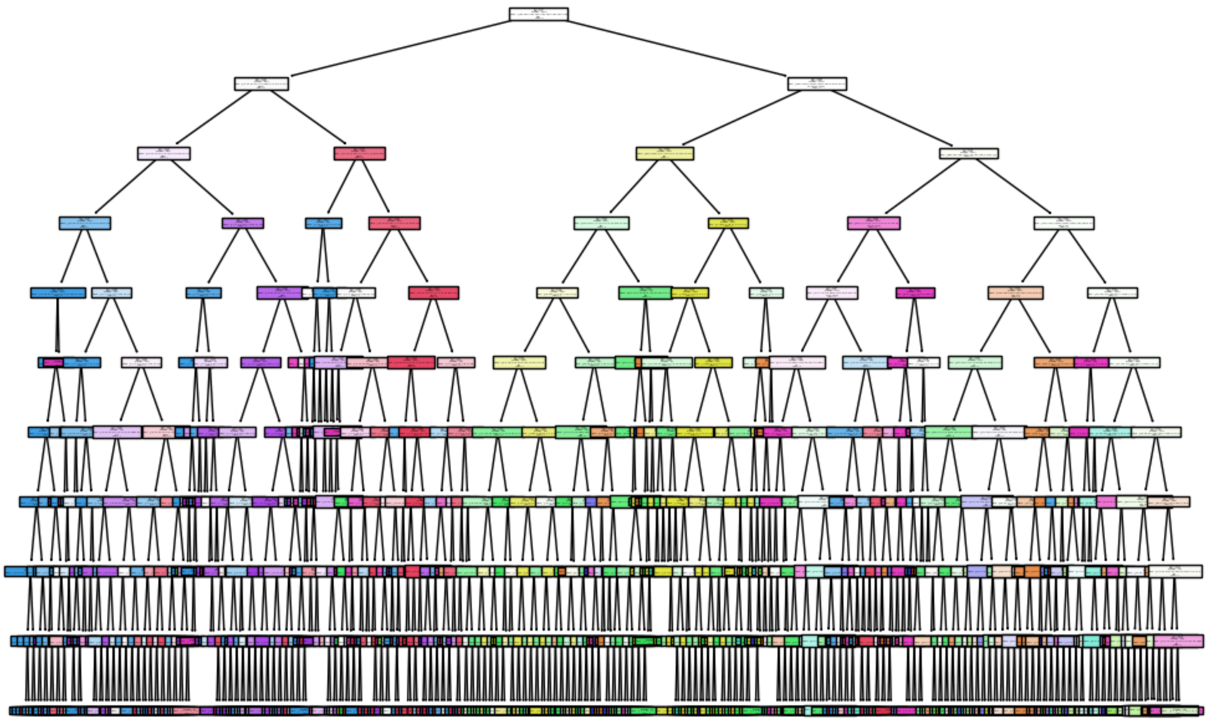
Implementation of a Decision Tree with:

- Max depth = 10

In [22]:

```
#################### Decision Tree ################################
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

# Decision Tree
decision_tree = DecisionTreeClassifier(max_depth=10)
decision_tree.fit(sampled_train_images_vectorized, sampled_train_labels)

# Evaluate the decision tree model on the testing set
decision_tree_accuracy = decision_tree.score(test_images_vectorized, test_labels)

# Print the accuracy of the decision tree model
print("Decision Tree Accuracy:", decision_tree_accuracy)


# Plot Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(decision_tree, filled=True, class_names=[str(i) for i in range(10)], feat
plt.show()
```

```
Decision Tree Accuracy: 0.7026
```



Step 5.2: Random Forest

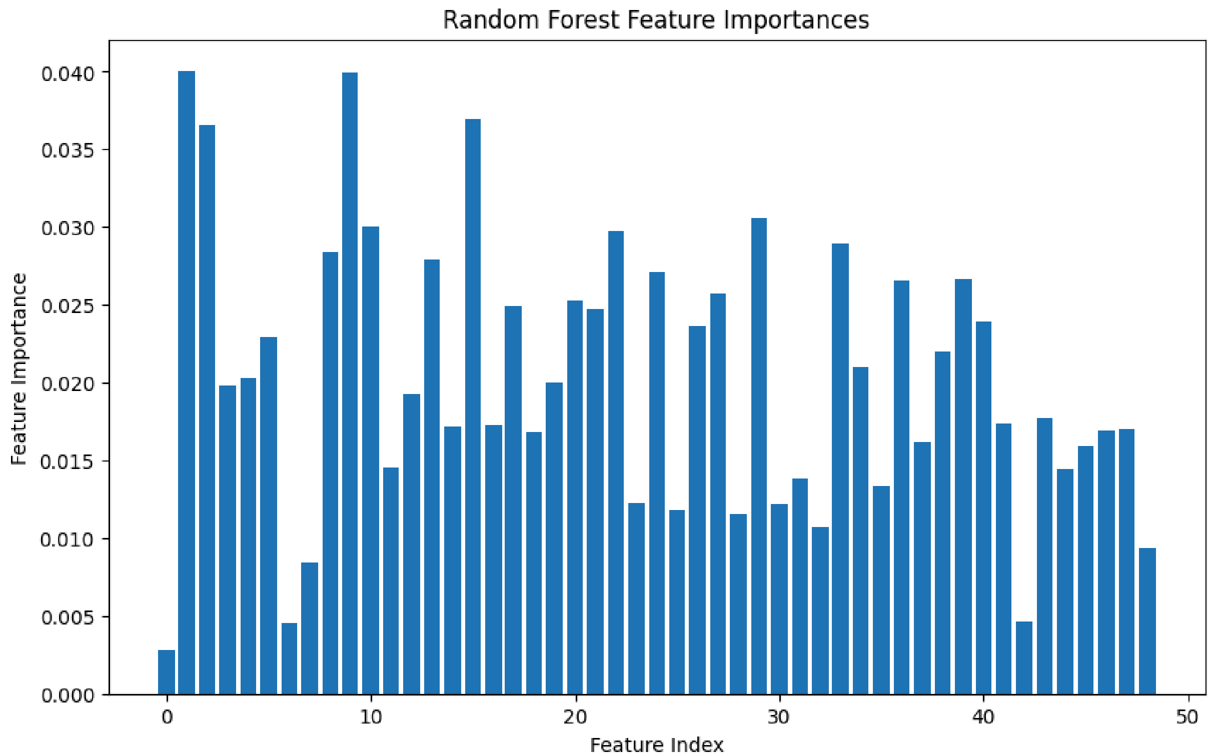
Implementation of Random Forest algorithm with:

- n = 100 estimators

In [23]:
```
# Random Forest
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(sampled_train_images_vectorized, sampled_train_labels)

# Evaluate the Random Forest model on the testing set
random_forest_accuracy = random_forest.score(test_images_vectorized, test_labels)

# Print the accuracy of the Random Forest model
print("Random Forest Accuracy:", random_forest_accuracy)


# Visualize Feature Importances for Random Forest
feature_importances = random_forest.feature_importances_
plt.figure(figsize=(10, 6))
plt.bar(range(len(feature_importances)), feature_importances)
plt.xlabel('Feature Index')
plt.ylabel('Feature Importance')
plt.title('Random Forest Feature Importances')
```

```
plt.show()
#################################################################################
```

```
Random Forest Accuracy: 0.8089
```



Step 6: Support Vector Machine (SVM)

- C = [1,10,100]
- Max iterations = 500
- Linear SVM
- SVM with RBF kernel : gamma = [0.02, 0.1, 1]
- Standarize data in order to achieve best performance results in regards to features domination(magnitude)
- (For instance many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the L1 and L2 regularizers of linear models) assume that all features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.): StandardScaler()
- Implementation of both algorithms

In [24]:

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.exceptions import ConvergenceWarning
import warnings

# Suppress convergence warnings
warnings.filterwarnings("ignore", category=ConvergenceWarning)

# Standardize the data
scaler = StandardScaler()
train_images_scaled = scaler.fit_transform(sampled_train_images_vectorized)
test_images_scaled = scaler.transform(test_images_vectorized)

# Different values of C
C_values = [1, 10, 100]

# Train and evaluate SVM classifiers
for C in C_values:
    # Linear SVM classifier
    linear_svm = SVC(kernel='linear', C=C, max_iter=500)
    linear_svm.fit(train_images_scaled, sampled_train_labels)  # Corrected the labe
    linear_accuracy = linear_svm.score(test_images_scaled, test_labels)
    print("Linear SVM with C =", C, "Accuracy:", linear_accuracy)

    # RBF kernel SVM classifier
    for gamma in [0.02, 0.1, 1]:
        rbf_svm = SVC(kernel='rbf', C=C, gamma=gamma, max_iter=500)
        rbf_svm.fit(train_images_scaled, sampled_train_labels)  # Corrected the lab
        rbf_accuracy = rbf_svm.score(test_images_scaled, test_labels)
        print("RBF SVM with C =", C, "and gamma =", gamma, "Accuracy:", rbf_accurac
```

```
Linear SVM with C = 1 Accuracy: 0.5216
RBF SVM with C = 1 and gamma = 0.02 Accuracy: 0.6683
RBF SVM with C = 1 and gamma = 0.1 Accuracy: 0.7772
RBF SVM with C = 1 and gamma = 1 Accuracy: 0.684
Linear SVM with C = 10 Accuracy: 0.3091
RBF SVM with C = 10 and gamma = 0.02 Accuracy: 0.6923
RBF SVM with C = 10 and gamma = 0.1 Accuracy: 0.758
RBF SVM with C = 10 and gamma = 1 Accuracy: 0.5242
Linear SVM with C = 100 Accuracy: 0.3952
RBF SVM with C = 100 and gamma = 0.02 Accuracy: 0.6684
RBF SVM with C = 100 and gamma = 0.1 Accuracy: 0.6851
RBF SVM with C = 100 and gamma = 1 Accuracy: 0.5394
```

Step 7: Feed Forward Neural Network

- Create a Sequential Model
- This sequential model consists of 3 layers (Dense func)
- Each layer has its own number of neurons
- On every hidden layer we use the LeakyRelU function (with a small negative slope)
- On the output layer we use the SoftMax function
- *Extra: On the first layer we need to specify the number of features (shape[1])

In [25]:
```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, LeakyReLU
from keras.optimizers import Adam
from keras.layers import Input
import matplotlib.pyplot as plt

# Define the structure of the model
model = Sequential([
    Input(shape=(sampled_train_images_vectorized.shape[1],)),
    Dense(100, activation=LeakyReLU(negative_slope=0.1)),
    Dense(100, activation=LeakyReLU(negative_slope=0.1)),
    Dense(50, activation=LeakyReLU(negative_slope=0.1)),
    Dense(10, activation='softmax')
])
```

Implmentation and Plot of the Sequential Model and Plot

- Batch size = 50 -> number of samples that will be passed to the network at the same time
- Epochs = 100 -> number of runs
- We want to print the last accuracy result of the model

In [26]:
```
# Total number of parameters in the model
num_parameters = model.count_params()
print("Total number of parameters in the model:", num_parameters)

# Define the loss function and optimizer
model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['a

# Train the model
history = model.fit(sampled_train_images_vectorized, sampled_train_labels, epochs=1


# Extract final accuracy from history object
final_train_accuracy = history.history['accuracy'][-1]
final_val_accuracy = history.history['val_accuracy'][-1]
```

```python
# Print the final accuracy
print("Final Training Accuracy:", final_train_accuracy)
print("Final Validation Accuracy:", final_val_accuracy)



# Plot the loss function and accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

plt.show()
```

```
Total number of parameters in the model: 20660
Epoch 1/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.5917 - loss: 1.0978 - val_acc
uracy: 0.7275 - val_loss: 0.7145
Epoch 2/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 887us/step - accuracy: 0.7436 - loss: 0.6690 - val_a
ccuracy: 0.7353 - val_loss: 0.6782
Epoch 3/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 885us/step - accuracy: 0.7581 - loss: 0.6254 - val_a
ccuracy: 0.7655 - val_loss: 0.6301
Epoch 4/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 883us/step - accuracy: 0.7614 - loss: 0.6112 - val_a
ccuracy: 0.7463 - val_loss: 0.6491
Epoch 5/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 900us/step - accuracy: 0.7719 - loss: 0.5925 - val_a
ccuracy: 0.7674 - val_loss: 0.6136
Epoch 6/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 876us/step - accuracy: 0.7821 - loss: 0.5673 - val_a
ccuracy: 0.7723 - val_loss: 0.5956
Epoch 7/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 887us/step - accuracy: 0.7902 - loss: 0.5514 - val_a
ccuracy: 0.7775 - val_loss: 0.5890
Epoch 8/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 889us/step - accuracy: 0.7958 - loss: 0.5371 - val_a
ccuracy: 0.7792 - val_loss: 0.5732
Epoch 9/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 975us/step - accuracy: 0.7984 - loss: 0.5261 - val_a
ccuracy: 0.7822 - val_loss: 0.5698
Epoch 10/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 876us/step - accuracy: 0.8060 - loss: 0.5146 - val_a
ccuracy: 0.7872 - val_loss: 0.5732
Epoch 11/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 906us/step - accuracy: 0.8026 - loss: 0.5161 - val_a
ccuracy: 0.7922 - val_loss: 0.5563
Epoch 12/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 945us/step - accuracy: 0.8041 - loss: 0.5075 - val_a
ccuracy: 0.7784 - val_loss: 0.5789
Epoch 13/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 955us/step - accuracy: 0.8084 - loss: 0.4939 - val_a
ccuracy: 0.7869 - val_loss: 0.5610
Epoch 14/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 933us/step - accuracy: 0.8105 - loss: 0.4958 - val_a
ccuracy: 0.7913 - val_loss: 0.5508
Epoch 15/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 943us/step - accuracy: 0.8122 - loss: 0.4859 - val_a
ccuracy: 0.7945 - val_loss: 0.5456
Epoch 16/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 931us/step - accuracy: 0.8158 - loss: 0.4765 - val_a
ccuracy: 0.7976 - val_loss: 0.5343
Epoch 17/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 944us/step - accuracy: 0.8199 - loss: 0.4690 - val_a
ccuracy: 0.7935 - val_loss: 0.5460
Epoch 18/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 928us/step - accuracy: 0.8174 - loss: 0.4708 - val_a
ccuracy: 0.8024 - val_loss: 0.5406
Epoch 19/100
```

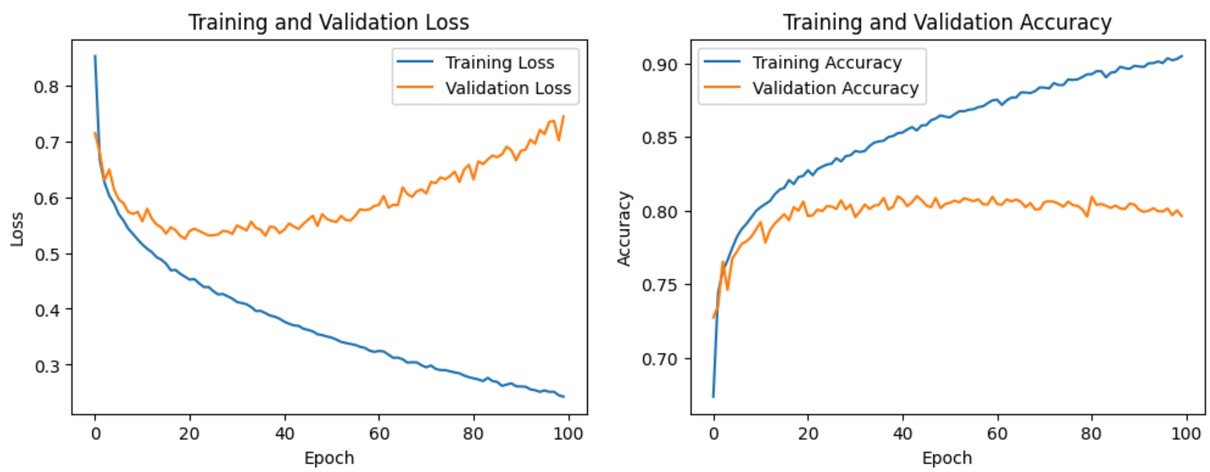
```
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 956us/step - accuracy: 0.8247 - loss: 0.4573 - val_a
ccuracy: 0.8000 - val_loss: 0.5305
Epoch 20/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 945us/step - accuracy: 0.8247 - loss: 0.4559 - val_a
ccuracy: 0.8063 - val_loss: 0.5251
Epoch 21/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 953us/step - accuracy: 0.8274 - loss: 0.4519 - val_a
ccuracy: 0.7963 - val_loss: 0.5394
Epoch 22/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 940us/step - accuracy: 0.8222 - loss: 0.4495 - val_a
ccuracy: 0.7969 - val_loss: 0.5434
Epoch 23/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8288 - loss: 0.4397 - val_acc
uracy: 0.8007 - val_loss: 0.5391
Epoch 24/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 940us/step - accuracy: 0.8298 - loss: 0.4351 - val_a
ccuracy: 0.7997 - val_loss: 0.5351
Epoch 25/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 934us/step - accuracy: 0.8339 - loss: 0.4343 - val_a
ccuracy: 0.8031 - val_loss: 0.5311
Epoch 26/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 972us/step - accuracy: 0.8345 - loss: 0.4260 - val_a
ccuracy: 0.8029 - val_loss: 0.5315
Epoch 27/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 930us/step - accuracy: 0.8380 - loss: 0.4199 - val_a
ccuracy: 0.8012 - val_loss: 0.5329
Epoch 28/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 938us/step - accuracy: 0.8362 - loss: 0.4191 - val_a
ccuracy: 0.8071 - val_loss: 0.5392
Epoch 29/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 940us/step - accuracy: 0.8377 - loss: 0.4177 - val_a
ccuracy: 0.8010 - val_loss: 0.5384
Epoch 30/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 938us/step - accuracy: 0.8389 - loss: 0.4117 - val_a
ccuracy: 0.8041 - val_loss: 0.5337
Epoch 31/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 946us/step - accuracy: 0.8408 - loss: 0.4117 - val_a
ccuracy: 0.7957 - val_loss: 0.5491
Epoch 32/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 949us/step - accuracy: 0.8383 - loss: 0.4101 - val_a
ccuracy: 0.7994 - val_loss: 0.5448
Epoch 33/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 945us/step - accuracy: 0.8415 - loss: 0.4093 - val_a
ccuracy: 0.8043 - val_loss: 0.5401
Epoch 34/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 948us/step - accuracy: 0.8438 - loss: 0.4018 - val_a
ccuracy: 0.8012 - val_loss: 0.5557
Epoch 35/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 951us/step - accuracy: 0.8459 - loss: 0.3926 - val_a
ccuracy: 0.8034 - val_loss: 0.5446
Epoch 36/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8524 - loss: 0.3856 - val_acc
uracy: 0.8040 - val_loss: 0.5412
Epoch 37/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 938us/step - accuracy: 0.8499 - loss: 0.3848 - val_a
ccuracy: 0.8087 - val_loss: 0.5307
```

```
Epoch 38/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 946us/step - accuracy: 0.8491 - loss: 0.3857 - val_a
ccuracy: 0.8008 - val_loss: 0.5471
Epoch 39/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 946us/step - accuracy: 0.8516 - loss: 0.3843 - val_a
ccuracy: 0.8027 - val_loss: 0.5456
Epoch 40/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 946us/step - accuracy: 0.8498 - loss: 0.3807 - val_a
ccuracy: 0.8098 - val_loss: 0.5353
Epoch 41/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 938us/step - accuracy: 0.8574 - loss: 0.3681 - val_a
ccuracy: 0.8076 - val_loss: 0.5416
Epoch 42/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 946us/step - accuracy: 0.8624 - loss: 0.3565 - val_a
ccuracy: 0.8031 - val_loss: 0.5525
Epoch 43/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 939us/step - accuracy: 0.8555 - loss: 0.3702 - val_a
ccuracy: 0.8058 - val_loss: 0.5473
Epoch 44/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 933us/step - accuracy: 0.8562 - loss: 0.3651 - val_a
ccuracy: 0.8099 - val_loss: 0.5427
Epoch 45/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 937us/step - accuracy: 0.8572 - loss: 0.3614 - val_a
ccuracy: 0.8073 - val_loss: 0.5518
Epoch 46/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 951us/step - accuracy: 0.8615 - loss: 0.3574 - val_a
ccuracy: 0.8036 - val_loss: 0.5591
Epoch 47/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 929us/step - accuracy: 0.8637 - loss: 0.3552 - val_a
ccuracy: 0.8024 - val_loss: 0.5666
Epoch 48/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 941us/step - accuracy: 0.8672 - loss: 0.3488 - val_a
ccuracy: 0.8086 - val_loss: 0.5483
Epoch 49/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 936us/step - accuracy: 0.8669 - loss: 0.3457 - val_a
ccuracy: 0.8018 - val_loss: 0.5692
Epoch 50/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 949us/step - accuracy: 0.8634 - loss: 0.3489 - val_a
ccuracy: 0.8045 - val_loss: 0.5607
Epoch 51/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 940us/step - accuracy: 0.8639 - loss: 0.3470 - val_a
ccuracy: 0.8054 - val_loss: 0.5573
Epoch 52/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 941us/step - accuracy: 0.8653 - loss: 0.3406 - val_a
ccuracy: 0.8068 - val_loss: 0.5552
Epoch 53/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 953us/step - accuracy: 0.8674 - loss: 0.3389 - val_a
ccuracy: 0.8057 - val_loss: 0.5663
Epoch 54/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 951us/step - accuracy: 0.8688 - loss: 0.3379 - val_a
ccuracy: 0.8085 - val_loss: 0.5587
Epoch 55/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 954us/step - accuracy: 0.8677 - loss: 0.3362 - val_a
ccuracy: 0.8075 - val_loss: 0.5580
Epoch 56/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 948us/step - accuracy: 0.8699 - loss: 0.3361 - val_a
```

```
ccuracy: 0.8065 - val_loss: 0.5664
Epoch 57/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 954us/step - accuracy: 0.8705 - loss: 0.3351 - val_a
ccuracy: 0.8078 - val_loss: 0.5780
Epoch 58/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8689 - loss: 0.3360 - val_acc
uracy: 0.8044 - val_loss: 0.5763
Epoch 59/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 947us/step - accuracy: 0.8785 - loss: 0.3169 - val_a
ccuracy: 0.8043 - val_loss: 0.5780
Epoch 60/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 978us/step - accuracy: 0.8805 - loss: 0.3117 - val_a
ccuracy: 0.8095 - val_loss: 0.5838
Epoch 61/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 949us/step - accuracy: 0.8796 - loss: 0.3126 - val_a
ccuracy: 0.8045 - val_loss: 0.5859
Epoch 62/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 940us/step - accuracy: 0.8744 - loss: 0.3187 - val_a
ccuracy: 0.8041 - val_loss: 0.6018
Epoch 63/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 955us/step - accuracy: 0.8768 - loss: 0.3172 - val_a
ccuracy: 0.8077 - val_loss: 0.5808
Epoch 64/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 945us/step - accuracy: 0.8808 - loss: 0.3025 - val_a
ccuracy: 0.8063 - val_loss: 0.5861
Epoch 65/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 952us/step - accuracy: 0.8777 - loss: 0.3094 - val_a
ccuracy: 0.8076 - val_loss: 0.5855
Epoch 66/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 956us/step - accuracy: 0.8862 - loss: 0.2975 - val_a
ccuracy: 0.8061 - val_loss: 0.6172
Epoch 67/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 954us/step - accuracy: 0.8822 - loss: 0.2993 - val_a
ccuracy: 0.8024 - val_loss: 0.6054
Epoch 68/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 952us/step - accuracy: 0.8797 - loss: 0.3002 - val_a
ccuracy: 0.8052 - val_loss: 0.6002
Epoch 69/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 939us/step - accuracy: 0.8859 - loss: 0.2913 - val_a
ccuracy: 0.8009 - val_loss: 0.6101
Epoch 70/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 969us/step - accuracy: 0.8846 - loss: 0.2953 - val_a
ccuracy: 0.8011 - val_loss: 0.6137
Epoch 71/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 946us/step - accuracy: 0.8854 - loss: 0.2935 - val_a
ccuracy: 0.8059 - val_loss: 0.6064
Epoch 72/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 959us/step - accuracy: 0.8844 - loss: 0.2959 - val_a
ccuracy: 0.8064 - val_loss: 0.6271
Epoch 73/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 949us/step - accuracy: 0.8905 - loss: 0.2853 - val_a
ccuracy: 0.8061 - val_loss: 0.6248
Epoch 74/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 939us/step - accuracy: 0.8915 - loss: 0.2755 - val_a
ccuracy: 0.8047 - val_loss: 0.6351
Epoch 75/100
```

```
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 945us/step - accuracy: 0.8866 - loss: 0.2845 - val_a
ccuracy: 0.8027 - val_loss: 0.6318
Epoch 76/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8905 - loss: 0.2860 - val_acc
uracy: 0.8062 - val_loss: 0.6372
Epoch 77/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 956us/step - accuracy: 0.8892 - loss: 0.2863 - val_a
ccuracy: 0.8038 - val_loss: 0.6455
Epoch 78/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 950us/step - accuracy: 0.8892 - loss: 0.2825 - val_a
ccuracy: 0.8033 - val_loss: 0.6269
Epoch 79/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 951us/step - accuracy: 0.8940 - loss: 0.2766 - val_a
ccuracy: 0.8009 - val_loss: 0.6490
Epoch 80/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 959us/step - accuracy: 0.8953 - loss: 0.2725 - val_a
ccuracy: 0.7960 - val_loss: 0.6576
Epoch 81/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 955us/step - accuracy: 0.8949 - loss: 0.2733 - val_a
ccuracy: 0.8095 - val_loss: 0.6316
Epoch 82/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 957us/step - accuracy: 0.8995 - loss: 0.2682 - val_a
ccuracy: 0.8040 - val_loss: 0.6638
Epoch 83/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 944us/step - accuracy: 0.8982 - loss: 0.2634 - val_a
ccuracy: 0.8046 - val_loss: 0.6589
Epoch 84/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 949us/step - accuracy: 0.8981 - loss: 0.2624 - val_a
ccuracy: 0.8033 - val_loss: 0.6670
Epoch 85/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 953us/step - accuracy: 0.8969 - loss: 0.2626 - val_a
ccuracy: 0.8019 - val_loss: 0.6742
Epoch 86/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 952us/step - accuracy: 0.8975 - loss: 0.2641 - val_a
ccuracy: 0.8035 - val_loss: 0.6712
Epoch 87/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 957us/step - accuracy: 0.9014 - loss: 0.2537 - val_a
ccuracy: 0.8016 - val_loss: 0.6762
Epoch 88/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 962us/step - accuracy: 0.8960 - loss: 0.2637 - val_a
ccuracy: 0.8011 - val_loss: 0.6900
Epoch 89/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 951us/step - accuracy: 0.8987 - loss: 0.2635 - val_a
ccuracy: 0.8048 - val_loss: 0.6833
Epoch 90/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9006 - loss: 0.2555 - val_acc
uracy: 0.8042 - val_loss: 0.6657
Epoch 91/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 974us/step - accuracy: 0.9019 - loss: 0.2560 - val_a
ccuracy: 0.8006 - val_loss: 0.6828
Epoch 92/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 952us/step - accuracy: 0.8990 - loss: 0.2572 - val_a
ccuracy: 0.7992 - val_loss: 0.6848
Epoch 93/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9003 - loss: 0.2491 - val_acc
uracy: 0.8000 - val_loss: 0.7028
```

```
Epoch 94/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 986us/step - accuracy: 0.9031 - loss: 0.2477 - val_a
ccuracy: 0.8017 - val_loss: 0.6954
Epoch 95/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 972us/step - accuracy: 0.9065 - loss: 0.2374 - val_a
ccuracy: 0.7997 - val_loss: 0.7202
Epoch 96/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 967us/step - accuracy: 0.9069 - loss: 0.2421 - val_a
ccuracy: 0.7995 - val_loss: 0.7124
Epoch 97/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 965us/step - accuracy: 0.9061 - loss: 0.2502 - val_a
ccuracy: 0.8015 - val_loss: 0.7344
Epoch 98/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 963us/step - accuracy: 0.9060 - loss: 0.2463 - val_a
ccuracy: 0.7970 - val_loss: 0.7359
Epoch 99/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 956us/step - accuracy: 0.9018 - loss: 0.2460 - val_a
ccuracy: 0.8003 - val_loss: 0.7012
Epoch 100/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 964us/step - accuracy: 0.9076 - loss: 0.2372 - val_a
ccuracy: 0.7964 - val_loss: 0.7440
Final Training Accuracy: 0.9049999713897705
Final Validation Accuracy: 0.7964000105857849
```



Step 8: Convolutional Neural Network (CNN)

- Version 1
- **2** Convolutional Layers
- Max Pooling
- Dense with Dropout
- Optimizer = Adam
- Loss function = 'Categorical Cross Entropy'

In [30]:
```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
```

```python
from keras.optimizers import Adam
import matplotlib.pyplot as plt

# Define the CNN model structure
# (We could just pass in the format 49x1 but we need to reshape it because we want
model = Sequential([
    Input(shape=(7, 7, 1)),  # Define input shape
    Conv2D(32, (3, 3), activation='relu', padding= 'same'),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    # Conv2D(128, (3, 3), activation='relu', padding= 'same'),  # Add another convo
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(100, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])


# Total number of parameters in the model
num_parameters = model.count_params()
print("Total number of parameters in the model:", num_parameters)

# Define the loss function and optimizer
model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['a

# Train the model (We could just pass in the format 49x1 but we need to reshape it
history = model.fit(sampled_train_images_vectorized.reshape(-1, 7, 7, 1), sampled_t

# Extract final accuracy from history object
final_training_accuracy = history.history['accuracy'][-1]
final_validation_accuracy = history.history['val_accuracy'][-1]

print("Final Training Accuracy:", final_training_accuracy)
print("Final Validation Accuracy:", final_validation_accuracy)


# Plot the loss function and accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

plt.show()
```

```
Total number of parameters in the model: 77526
Epoch 1/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5318 - loss: 1.2301 - val_acc
uracy: 0.7355 - val_loss: 0.6919
Epoch 2/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7279 - loss: 0.7172 - val_acc
uracy: 0.7668 - val_loss: 0.6182
Epoch 3/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7604 - loss: 0.6352 - val_acc
uracy: 0.7832 - val_loss: 0.5775
Epoch 4/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7835 - loss: 0.5800 - val_acc
uracy: 0.7917 - val_loss: 0.5546
Epoch 5/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7952 - loss: 0.5475 - val_acc
uracy: 0.7970 - val_loss: 0.5362
Epoch 6/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7984 - loss: 0.5335 - val_acc
uracy: 0.8025 - val_loss: 0.5268
Epoch 7/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8025 - loss: 0.5212 - val_acc
uracy: 0.8062 - val_loss: 0.5195
Epoch 8/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8117 - loss: 0.5017 - val_acc
uracy: 0.8111 - val_loss: 0.5014
Epoch 9/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8155 - loss: 0.4877 - val_acc
uracy: 0.8095 - val_loss: 0.4980
Epoch 10/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8242 - loss: 0.4704 - val_acc
uracy: 0.8119 - val_loss: 0.5061
Epoch 11/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8280 - loss: 0.4642 - val_acc
uracy: 0.8142 - val_loss: 0.5040
Epoch 12/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8241 - loss: 0.4639 - val_acc
uracy: 0.8169 - val_loss: 0.4851
Epoch 13/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8312 - loss: 0.4462 - val_acc
uracy: 0.8181 - val_loss: 0.4856
Epoch 14/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8328 - loss: 0.4445 - val_acc
uracy: 0.8224 - val_loss: 0.4919
Epoch 15/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8336 - loss: 0.4363 - val_acc
uracy: 0.8225 - val_loss: 0.4772
Epoch 16/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8390 - loss: 0.4176 - val_acc
uracy: 0.8245 - val_loss: 0.4826
Epoch 17/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8455 - loss: 0.4120 - val_acc
uracy: 0.8203 - val_loss: 0.4970
Epoch 18/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8444 - loss: 0.4079 - val_acc
uracy: 0.8216 - val_loss: 0.4866
Epoch 19/100
```

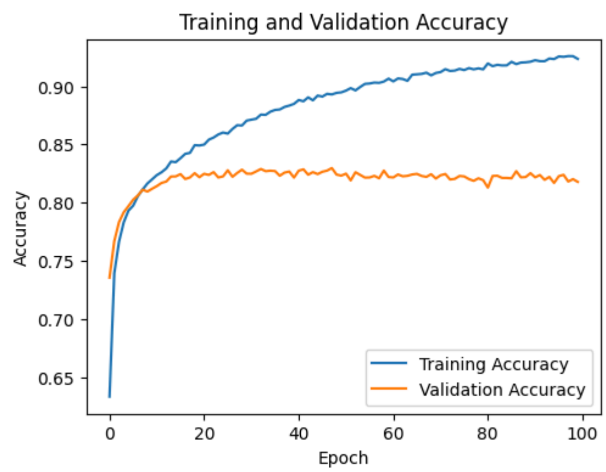
```
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8495 - loss: 0.4006 - val_acc
uracy: 0.8253 - val_loss: 0.4849
Epoch 20/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8503 - loss: 0.3906 - val_acc
uracy: 0.8218 - val_loss: 0.4815
Epoch 21/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8504 - loss: 0.3954 - val_acc
uracy: 0.8248 - val_loss: 0.4828
Epoch 22/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8548 - loss: 0.3777 - val_acc
uracy: 0.8238 - val_loss: 0.4822
Epoch 23/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8600 - loss: 0.3754 - val_acc
uracy: 0.8262 - val_loss: 0.4888
Epoch 24/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8627 - loss: 0.3607 - val_acc
uracy: 0.8216 - val_loss: 0.4921
Epoch 25/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8625 - loss: 0.3587 - val_acc
uracy: 0.8225 - val_loss: 0.4850
Epoch 26/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8620 - loss: 0.3547 - val_acc
uracy: 0.8278 - val_loss: 0.4896
Epoch 27/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8646 - loss: 0.3575 - val_acc
uracy: 0.8222 - val_loss: 0.5088
Epoch 28/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8661 - loss: 0.3487 - val_acc
uracy: 0.8255 - val_loss: 0.4964
Epoch 29/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8654 - loss: 0.3429 - val_acc
uracy: 0.8283 - val_loss: 0.4905
Epoch 30/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8745 - loss: 0.3280 - val_acc
uracy: 0.8249 - val_loss: 0.5057
Epoch 31/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8759 - loss: 0.3243 - val_acc
uracy: 0.8248 - val_loss: 0.5082
Epoch 32/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8736 - loss: 0.3275 - val_acc
uracy: 0.8268 - val_loss: 0.5138
Epoch 33/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8766 - loss: 0.3259 - val_acc
uracy: 0.8289 - val_loss: 0.5091
Epoch 34/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8801 - loss: 0.3122 - val_acc
uracy: 0.8270 - val_loss: 0.5240
Epoch 35/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8838 - loss: 0.3036 - val_acc
uracy: 0.8274 - val_loss: 0.5255
Epoch 36/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8825 - loss: 0.3048 - val_acc
uracy: 0.8271 - val_loss: 0.5186
Epoch 37/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8818 - loss: 0.3082 - val_acc
uracy: 0.8231 - val_loss: 0.5249
```

```
Epoch 38/100
600/600  2s 3ms/step - accuracy: 0.8854 - loss: 0.2914 - val_acc
uracy: 0.8258 - val_loss: 0.5266
Epoch 39/100
600/600  2s 3ms/step - accuracy: 0.8842 - loss: 0.2936 - val_acc
uracy: 0.8267 - val_loss: 0.5541
Epoch 40/100
600/600  2s 3ms/step - accuracy: 0.8875 - loss: 0.2889 - val_acc
uracy: 0.8215 - val_loss: 0.5634
Epoch 41/100
600/600  2s 3ms/step - accuracy: 0.8923 - loss: 0.2794 - val_acc
uracy: 0.8274 - val_loss: 0.5455
Epoch 42/100
600/600  2s 3ms/step - accuracy: 0.8881 - loss: 0.2850 - val_acc
uracy: 0.8286 - val_loss: 0.5657
Epoch 43/100
600/600  2s 3ms/step - accuracy: 0.8891 - loss: 0.2817 - val_acc
uracy: 0.8240 - val_loss: 0.5594
Epoch 44/100
600/600  2s 3ms/step - accuracy: 0.8927 - loss: 0.2715 - val_acc
uracy: 0.8262 - val_loss: 0.5691
Epoch 45/100
600/600  2s 3ms/step - accuracy: 0.8909 - loss: 0.2751 - val_acc
uracy: 0.8245 - val_loss: 0.5637
Epoch 46/100
600/600  2s 3ms/step - accuracy: 0.8977 - loss: 0.2634 - val_acc
uracy: 0.8265 - val_loss: 0.5587
Epoch 47/100
600/600  2s 3ms/step - accuracy: 0.8964 - loss: 0.2663 - val_acc
uracy: 0.8275 - val_loss: 0.5523
Epoch 48/100
600/600  2s 3ms/step - accuracy: 0.8967 - loss: 0.2590 - val_acc
uracy: 0.8297 - val_loss: 0.5821
Epoch 49/100
600/600  2s 4ms/step - accuracy: 0.8945 - loss: 0.2700 - val_acc
uracy: 0.8241 - val_loss: 0.6093
Epoch 50/100
600/600  2s 4ms/step - accuracy: 0.8925 - loss: 0.2659 - val_acc
uracy: 0.8232 - val_loss: 0.5956
Epoch 51/100
600/600  2s 4ms/step - accuracy: 0.9001 - loss: 0.2573 - val_acc
uracy: 0.8249 - val_loss: 0.5973
Epoch 52/100
600/600  2s 4ms/step - accuracy: 0.9033 - loss: 0.2494 - val_acc
uracy: 0.8191 - val_loss: 0.6157
Epoch 53/100
600/600  2s 3ms/step - accuracy: 0.9002 - loss: 0.2520 - val_acc
uracy: 0.8261 - val_loss: 0.6009
Epoch 54/100
600/600  2s 3ms/step - accuracy: 0.9015 - loss: 0.2548 - val_acc
uracy: 0.8238 - val_loss: 0.6156
Epoch 55/100
600/600  2s 3ms/step - accuracy: 0.9048 - loss: 0.2432 - val_acc
uracy: 0.8215 - val_loss: 0.6372
Epoch 56/100
600/600  2s 3ms/step - accuracy: 0.9044 - loss: 0.2394 - val_acc
```

```
uracy: 0.8216 - val_loss: 0.6393
Epoch 57/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9049 - loss: 0.2454 - val_acc
uracy: 0.8230 - val_loss: 0.6251
Epoch 58/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9032 - loss: 0.2396 - val_acc
uracy: 0.8209 - val_loss: 0.6455
Epoch 59/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9050 - loss: 0.2411 - val_acc
uracy: 0.8275 - val_loss: 0.6186
Epoch 60/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9097 - loss: 0.2310 - val_acc
uracy: 0.8220 - val_loss: 0.6718
Epoch 61/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9038 - loss: 0.2367 - val_acc
uracy: 0.8216 - val_loss: 0.6488
Epoch 62/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9135 - loss: 0.2265 - val_acc
uracy: 0.8242 - val_loss: 0.6570
Epoch 63/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9067 - loss: 0.2280 - val_acc
uracy: 0.8226 - val_loss: 0.6708
Epoch 64/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9050 - loss: 0.2306 - val_acc
uracy: 0.8230 - val_loss: 0.6727
Epoch 65/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9139 - loss: 0.2223 - val_acc
uracy: 0.8242 - val_loss: 0.6906
Epoch 66/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9105 - loss: 0.2265 - val_acc
uracy: 0.8247 - val_loss: 0.6761
Epoch 67/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9114 - loss: 0.2172 - val_acc
uracy: 0.8229 - val_loss: 0.6868
Epoch 68/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9159 - loss: 0.2199 - val_acc
uracy: 0.8221 - val_loss: 0.6782
Epoch 69/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9084 - loss: 0.2255 - val_acc
uracy: 0.8244 - val_loss: 0.6728
Epoch 70/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9144 - loss: 0.2101 - val_acc
uracy: 0.8207 - val_loss: 0.7200
Epoch 71/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9151 - loss: 0.2148 - val_acc
uracy: 0.8235 - val_loss: 0.7092
Epoch 72/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9169 - loss: 0.2117 - val_acc
uracy: 0.8246 - val_loss: 0.7125
Epoch 73/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9182 - loss: 0.2074 - val_acc
uracy: 0.8199 - val_loss: 0.6876
Epoch 74/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9150 - loss: 0.2121 - val_acc
uracy: 0.8204 - val_loss: 0.7265
Epoch 75/100
```

```
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9125 - loss: 0.2177 - val_acc
uracy: 0.8229 - val_loss: 0.7599
Epoch 76/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9165 - loss: 0.2039 - val_acc
uracy: 0.8225 - val_loss: 0.7393
Epoch 77/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9111 - loss: 0.2160 - val_acc
uracy: 0.8202 - val_loss: 0.7499
Epoch 78/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9160 - loss: 0.2052 - val_acc
uracy: 0.8187 - val_loss: 0.7648
Epoch 79/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9182 - loss: 0.2012 - val_acc
uracy: 0.8209 - val_loss: 0.7466
Epoch 80/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9172 - loss: 0.2037 - val_acc
uracy: 0.8193 - val_loss: 0.7443
Epoch 81/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9214 - loss: 0.1962 - val_acc
uracy: 0.8129 - val_loss: 0.7898
Epoch 82/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9194 - loss: 0.1974 - val_acc
uracy: 0.8229 - val_loss: 0.7629
Epoch 83/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9188 - loss: 0.2014 - val_acc
uracy: 0.8231 - val_loss: 0.7943
Epoch 84/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9198 - loss: 0.1969 - val_acc
uracy: 0.8212 - val_loss: 0.7976
Epoch 85/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9182 - loss: 0.1957 - val_acc
uracy: 0.8212 - val_loss: 0.8014
Epoch 86/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9223 - loss: 0.1989 - val_acc
uracy: 0.8209 - val_loss: 0.8292
Epoch 87/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9225 - loss: 0.1940 - val_acc
uracy: 0.8270 - val_loss: 0.7948
Epoch 88/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9189 - loss: 0.1949 - val_acc
uracy: 0.8217 - val_loss: 0.8142
Epoch 89/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9215 - loss: 0.1909 - val_acc
uracy: 0.8220 - val_loss: 0.8383
Epoch 90/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9209 - loss: 0.1932 - val_acc
uracy: 0.8253 - val_loss: 0.8316
Epoch 91/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9247 - loss: 0.1861 - val_acc
uracy: 0.8217 - val_loss: 0.8438
Epoch 92/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9252 - loss: 0.1851 - val_acc
uracy: 0.8237 - val_loss: 0.8185
Epoch 93/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9217 - loss: 0.1891 - val_acc
uracy: 0.8196 - val_loss: 0.8261
```

```
Epoch 94/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9238 - loss: 0.1856 - val_acc
uracy: 0.8221 - val_loss: 0.8516
Epoch 95/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9255 - loss: 0.1817 - val_acc
uracy: 0.8169 - val_loss: 0.8429
Epoch 96/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9280 - loss: 0.1803 - val_acc
uracy: 0.8227 - val_loss: 0.8233
Epoch 97/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9262 - loss: 0.1835 - val_acc
uracy: 0.8239 - val_loss: 0.8524
Epoch 98/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9270 - loss: 0.1841 - val_acc
uracy: 0.8181 - val_loss: 0.8648
Epoch 99/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9256 - loss: 0.1790 - val_acc
uracy: 0.8202 - val_loss: 0.8832
Epoch 100/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9256 - loss: 0.1865 - val_acc
uracy: 0.8178 - val_loss: 0.8755
Final Training Accuracy: 0.9235666394233704
Final Validation Accuracy: 0.817799985408783
```



Step 8: Convolutional Neural Network (CNN)

- Version 2
- **3** Convolutional Layers
- Max Pooling
- Dense with Dropout
- Optimizer = Adam
- Loss function = 'Categorical Cross Entropy'

In [33]:
```python
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
```

```
from keras.optimizers import Adam
import matplotlib.pyplot as plt

# Define the CNN model structure
# (We could just pass in the format 49x1 but we need to reshape it because we want
model = Sequential([
    Input(shape=(7, 7, 1)),  # Define input shape
    Conv2D(32, (3, 3), activation='relu', padding= 'same'),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    Conv2D(128, (3, 3), activation='relu', padding= 'same'),  # Add another convolu
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(50, activation='relu'),
    Dropout(0.15),
    Dense(10, activation='softmax')
])


# Total number of parameters in the model
num_parameters = model.count_params()
print("Total number of parameters in the model:", num_parameters)

# Define the loss function and optimizer
model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['a

# Train the model (We could just pass in the format 49x1 but we need to reshape it
history = model.fit(sampled_train_images_vectorized.reshape(-1, 7, 7, 1), sampled_t

# Extract final accuracy from history object
final_training_accuracy = history.history['accuracy'][-1]
final_validation_accuracy = history.history['val_accuracy'][-1]

print("Final Training Accuracy:", final_training_accuracy)
print("Final Validation Accuracy:", final_validation_accuracy)


# Plot the loss function and accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

plt.show()
```

```
Total number of parameters in the model: 150832
Epoch 1/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.5398 - loss: 1.2024 - val_acc
uracy: 0.7282 - val_loss: 0.7000
Epoch 2/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7395 - loss: 0.6888 - val_acc
uracy: 0.7755 - val_loss: 0.5887
Epoch 3/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.7719 - loss: 0.6015 - val_acc
uracy: 0.7857 - val_loss: 0.5690
Epoch 4/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.7904 - loss: 0.5559 - val_acc
uracy: 0.8049 - val_loss: 0.5279
Epoch 5/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8036 - loss: 0.5243 - val_acc
uracy: 0.8056 - val_loss: 0.5170
Epoch 6/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8099 - loss: 0.5023 - val_acc
uracy: 0.8137 - val_loss: 0.5172
Epoch 7/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8239 - loss: 0.4646 - val_acc
uracy: 0.8142 - val_loss: 0.5001
Epoch 8/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8271 - loss: 0.4557 - val_acc
uracy: 0.8220 - val_loss: 0.4877
Epoch 9/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8351 - loss: 0.4359 - val_acc
uracy: 0.8214 - val_loss: 0.4877
Epoch 10/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8440 - loss: 0.4176 - val_acc
uracy: 0.8180 - val_loss: 0.4970
Epoch 11/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8506 - loss: 0.3958 - val_acc
uracy: 0.8223 - val_loss: 0.5032
Epoch 12/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8563 - loss: 0.3819 - val_acc
uracy: 0.8252 - val_loss: 0.4969
Epoch 13/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8581 - loss: 0.3743 - val_acc
uracy: 0.8265 - val_loss: 0.5035
Epoch 14/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8642 - loss: 0.3542 - val_acc
uracy: 0.8208 - val_loss: 0.5290
Epoch 15/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8653 - loss: 0.3467 - val_acc
uracy: 0.8241 - val_loss: 0.5103
Epoch 16/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8775 - loss: 0.3243 - val_acc
uracy: 0.8251 - val_loss: 0.5124
Epoch 17/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8818 - loss: 0.3113 - val_acc
uracy: 0.8159 - val_loss: 0.5304
Epoch 18/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8806 - loss: 0.3104 - val_acc
uracy: 0.8257 - val_loss: 0.5420
Epoch 19/100
```

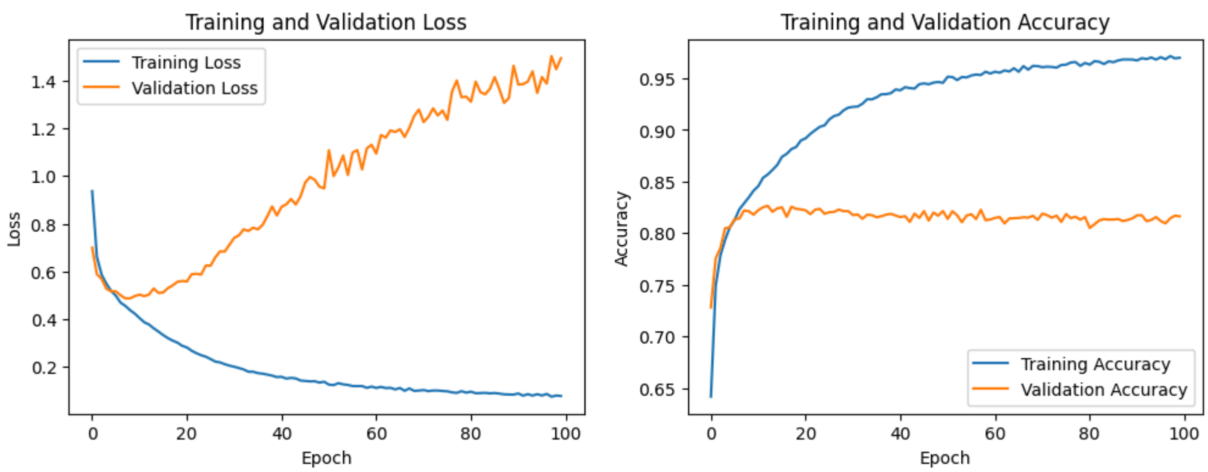
```
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8874 - loss: 0.2916 - val_acc
uracy: 0.8236 - val_loss: 0.5568
Epoch 20/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8895 - loss: 0.2891 - val_acc
uracy: 0.8231 - val_loss: 0.5606
Epoch 21/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8960 - loss: 0.2725 - val_acc
uracy: 0.8222 - val_loss: 0.5583
Epoch 22/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8965 - loss: 0.2651 - val_acc
uracy: 0.8185 - val_loss: 0.5885
Epoch 23/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9018 - loss: 0.2503 - val_acc
uracy: 0.8228 - val_loss: 0.5906
Epoch 24/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9064 - loss: 0.2395 - val_acc
uracy: 0.8236 - val_loss: 0.5871
Epoch 25/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9080 - loss: 0.2350 - val_acc
uracy: 0.8189 - val_loss: 0.6249
Epoch 26/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9156 - loss: 0.2204 - val_acc
uracy: 0.8203 - val_loss: 0.6240
Epoch 27/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9170 - loss: 0.2153 - val_acc
uracy: 0.8207 - val_loss: 0.6594
Epoch 28/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9183 - loss: 0.2097 - val_acc
uracy: 0.8229 - val_loss: 0.6844
Epoch 29/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9254 - loss: 0.1991 - val_acc
uracy: 0.8214 - val_loss: 0.6837
Epoch 30/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9237 - loss: 0.1987 - val_acc
uracy: 0.8215 - val_loss: 0.7127
Epoch 31/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9255 - loss: 0.1926 - val_acc
uracy: 0.8176 - val_loss: 0.7408
Epoch 32/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9263 - loss: 0.1862 - val_acc
uracy: 0.8182 - val_loss: 0.7518
Epoch 33/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9273 - loss: 0.1816 - val_acc
uracy: 0.8140 - val_loss: 0.7768
Epoch 34/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9333 - loss: 0.1709 - val_acc
uracy: 0.8180 - val_loss: 0.7699
Epoch 35/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9294 - loss: 0.1796 - val_acc
uracy: 0.8175 - val_loss: 0.7839
Epoch 36/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9327 - loss: 0.1694 - val_acc
uracy: 0.8155 - val_loss: 0.7767
Epoch 37/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9380 - loss: 0.1654 - val_acc
uracy: 0.8165 - val_loss: 0.7958
```

```
Epoch 38/100
600/600 4s 6ms/step - accuracy: 0.9372 - loss: 0.1620 - val_acc
uracy: 0.8180 - val_loss: 0.8320
Epoch 39/100
600/600 4s 6ms/step - accuracy: 0.9385 - loss: 0.1591 - val_acc
uracy: 0.8188 - val_loss: 0.8727
Epoch 40/100
600/600 4s 6ms/step - accuracy: 0.9393 - loss: 0.1544 - val_acc
uracy: 0.8172 - val_loss: 0.8349
Epoch 41/100
600/600 4s 7ms/step - accuracy: 0.9368 - loss: 0.1612 - val_acc
uracy: 0.8156 - val_loss: 0.8712
Epoch 42/100
600/600 4s 6ms/step - accuracy: 0.9425 - loss: 0.1472 - val_acc
uracy: 0.8166 - val_loss: 0.8814
Epoch 43/100
600/600 4s 6ms/step - accuracy: 0.9414 - loss: 0.1553 - val_acc
uracy: 0.8110 - val_loss: 0.9039
Epoch 44/100
600/600 4s 7ms/step - accuracy: 0.9437 - loss: 0.1416 - val_acc
uracy: 0.8190 - val_loss: 0.8809
Epoch 45/100
600/600 4s 7ms/step - accuracy: 0.9478 - loss: 0.1350 - val_acc
uracy: 0.8148 - val_loss: 0.9143
Epoch 46/100
600/600 4s 7ms/step - accuracy: 0.9460 - loss: 0.1360 - val_acc
uracy: 0.8214 - val_loss: 0.9730
Epoch 47/100
600/600 4s 6ms/step - accuracy: 0.9423 - loss: 0.1408 - val_acc
uracy: 0.8123 - val_loss: 0.9962
Epoch 48/100
600/600 4s 6ms/step - accuracy: 0.9473 - loss: 0.1352 - val_acc
uracy: 0.8200 - val_loss: 0.9842
Epoch 49/100
600/600 4s 6ms/step - accuracy: 0.9481 - loss: 0.1293 - val_acc
uracy: 0.8170 - val_loss: 0.9555
Epoch 50/100
600/600 4s 6ms/step - accuracy: 0.9488 - loss: 0.1292 - val_acc
uracy: 0.8215 - val_loss: 0.9482
Epoch 51/100
600/600 4s 6ms/step - accuracy: 0.9534 - loss: 0.1225 - val_acc
uracy: 0.8139 - val_loss: 1.1082
Epoch 52/100
600/600 4s 6ms/step - accuracy: 0.9517 - loss: 0.1190 - val_acc
uracy: 0.8213 - val_loss: 1.0001
Epoch 53/100
600/600 4s 6ms/step - accuracy: 0.9508 - loss: 0.1272 - val_acc
uracy: 0.8173 - val_loss: 1.0362
Epoch 54/100
600/600 4s 6ms/step - accuracy: 0.9517 - loss: 0.1267 - val_acc
uracy: 0.8107 - val_loss: 1.0861
Epoch 55/100
600/600 4s 6ms/step - accuracy: 0.9558 - loss: 0.1137 - val_acc
uracy: 0.8172 - val_loss: 1.0046
Epoch 56/100
600/600 4s 6ms/step - accuracy: 0.9541 - loss: 0.1139 - val_acc
```

```
uracy: 0.8184 - val_loss: 1.0984
Epoch 57/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9552 - loss: 0.1185 - val_acc
uracy: 0.8138 - val_loss: 1.1086
Epoch 58/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9542 - loss: 0.1170 - val_acc
uracy: 0.8227 - val_loss: 1.0277
Epoch 59/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9592 - loss: 0.1069 - val_acc
uracy: 0.8126 - val_loss: 1.1160
Epoch 60/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9546 - loss: 0.1162 - val_acc
uracy: 0.8133 - val_loss: 1.1311
Epoch 61/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9598 - loss: 0.1047 - val_acc
uracy: 0.8149 - val_loss: 1.0942
Epoch 62/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9573 - loss: 0.1132 - val_acc
uracy: 0.8157 - val_loss: 1.1721
Epoch 63/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9612 - loss: 0.1042 - val_acc
uracy: 0.8095 - val_loss: 1.1610
Epoch 64/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9572 - loss: 0.1119 - val_acc
uracy: 0.8143 - val_loss: 1.1913
Epoch 65/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9633 - loss: 0.0963 - val_acc
uracy: 0.8146 - val_loss: 1.1846
Epoch 66/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9582 - loss: 0.1048 - val_acc
uracy: 0.8147 - val_loss: 1.1961
Epoch 67/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9623 - loss: 0.0996 - val_acc
uracy: 0.8155 - val_loss: 1.1639
Epoch 68/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9594 - loss: 0.1073 - val_acc
uracy: 0.8151 - val_loss: 1.2030
Epoch 69/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9637 - loss: 0.0972 - val_acc
uracy: 0.8169 - val_loss: 1.2515
Epoch 70/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9642 - loss: 0.0945 - val_acc
uracy: 0.8130 - val_loss: 1.2788
Epoch 71/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9609 - loss: 0.1030 - val_acc
uracy: 0.8190 - val_loss: 1.2263
Epoch 72/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9606 - loss: 0.1025 - val_acc
uracy: 0.8175 - val_loss: 1.2478
Epoch 73/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9633 - loss: 0.0949 - val_acc
uracy: 0.8151 - val_loss: 1.2836
Epoch 74/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9626 - loss: 0.0937 - val_acc
uracy: 0.8170 - val_loss: 1.2538
Epoch 75/100
```

```
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9627 - loss: 0.0975 - val_acc
uracy: 0.8110 - val_loss: 1.2746
Epoch 76/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9630 - loss: 0.0987 - val_acc
uracy: 0.8186 - val_loss: 1.2356
Epoch 77/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9664 - loss: 0.0928 - val_acc
uracy: 0.8146 - val_loss: 1.3539
Epoch 78/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9672 - loss: 0.0881 - val_acc
uracy: 0.8160 - val_loss: 1.4004
Epoch 79/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9635 - loss: 0.0941 - val_acc
uracy: 0.8132 - val_loss: 1.3304
Epoch 80/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9657 - loss: 0.0912 - val_acc
uracy: 0.8157 - val_loss: 1.3327
Epoch 81/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9645 - loss: 0.0919 - val_acc
uracy: 0.8051 - val_loss: 1.3116
Epoch 82/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9652 - loss: 0.0922 - val_acc
uracy: 0.8087 - val_loss: 1.3955
Epoch 83/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9674 - loss: 0.0875 - val_acc
uracy: 0.8124 - val_loss: 1.3515
Epoch 84/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9651 - loss: 0.0902 - val_acc
uracy: 0.8137 - val_loss: 1.3427
Epoch 85/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9661 - loss: 0.0868 - val_acc
uracy: 0.8132 - val_loss: 1.3654
Epoch 86/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9651 - loss: 0.0943 - val_acc
uracy: 0.8133 - val_loss: 1.4146
Epoch 87/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9694 - loss: 0.0839 - val_acc
uracy: 0.8138 - val_loss: 1.3640
Epoch 88/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9684 - loss: 0.0850 - val_acc
uracy: 0.8118 - val_loss: 1.3067
Epoch 89/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9689 - loss: 0.0789 - val_acc
uracy: 0.8129 - val_loss: 1.3264
Epoch 90/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9687 - loss: 0.0812 - val_acc
uracy: 0.8144 - val_loss: 1.4622
Epoch 91/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9681 - loss: 0.0888 - val_acc
uracy: 0.8174 - val_loss: 1.3839
Epoch 92/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9684 - loss: 0.0825 - val_acc
uracy: 0.8174 - val_loss: 1.3850
Epoch 93/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9710 - loss: 0.0805 - val_acc
uracy: 0.8117 - val_loss: 1.3959
```

```
Epoch 94/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9730 - loss: 0.0736 - val_acc
uracy: 0.8129 - val_loss: 1.4388
Epoch 95/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9685 - loss: 0.0848 - val_acc
uracy: 0.8157 - val_loss: 1.3481
Epoch 96/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9733 - loss: 0.0734 - val_acc
uracy: 0.8118 - val_loss: 1.4149
Epoch 97/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9712 - loss: 0.0799 - val_acc
uracy: 0.8096 - val_loss: 1.3876
Epoch 98/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9720 - loss: 0.0771 - val_acc
uracy: 0.8144 - val_loss: 1.5026
Epoch 99/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9696 - loss: 0.0795 - val_acc
uracy: 0.8170 - val_loss: 1.4478
Epoch 100/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9719 - loss: 0.0748 - val_acc
uracy: 0.8164 - val_loss: 1.4936
Final Training Accuracy: 0.9699000120162964
Final Validation Accuracy: 0.8163999915122986
```



Step 8: Convolutional Neural Network (CNN)

- Version 3
- **2** Convolutional Layers
- Max Pooling
- Dense with Dropout
- Optimizer = Adam
- Loss function = 'Categorical Cross Entropy'

In [35]:
```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
```

```python
from keras.optimizers import Adam
import matplotlib.pyplot as plt

# Define the CNN model structure
# (We could just pass in the format 49x1 but we need to reshape it because we want
model = Sequential([
    Input(shape=(7, 7, 1)),  # Define input shape
    Conv2D(32, (3, 3), activation='relu', padding= 'same'),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    # Conv2D(128, (3, 3), activation='relu', padding= 'same'),  # Add another convo
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(50, activation='relu'),
    Dropout(0.4),
    Dense(10, activation='softmax')
])


# Total number of parameters in the model
num_parameters = model.count_params()
print("Total number of parameters in the model:", num_parameters)

# Define the loss function and optimizer
model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['a

# Train the model (We could just pass in the format 49x1 but we need to reshape it
history = model.fit(sampled_train_images_vectorized.reshape(-1, 7, 7, 1), sampled_t

# Extract final accuracy from history object
final_training_accuracy = history.history['accuracy'][-1]
final_validation_accuracy = history.history['val_accuracy'][-1]

print("Final Training Accuracy:", final_training_accuracy)
print("Final Validation Accuracy:", final_validation_accuracy)


# Plot the loss function and accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

plt.show()
```

```
Total number of parameters in the model: 48176
Epoch 1/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.4792 - loss: 1.3917 - val_acc
uracy: 0.7129 - val_loss: 0.7585
Epoch 2/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6976 - loss: 0.8225 - val_acc
uracy: 0.7423 - val_loss: 0.6698
Epoch 3/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7247 - loss: 0.7393 - val_acc
uracy: 0.7573 - val_loss: 0.6357
Epoch 4/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7382 - loss: 0.6994 - val_acc
uracy: 0.7815 - val_loss: 0.5934
Epoch 5/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7528 - loss: 0.6643 - val_acc
uracy: 0.7849 - val_loss: 0.5741
Epoch 6/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7651 - loss: 0.6236 - val_acc
uracy: 0.7866 - val_loss: 0.5627
Epoch 7/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7655 - loss: 0.6195 - val_acc
uracy: 0.7881 - val_loss: 0.5573
Epoch 8/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7756 - loss: 0.5987 - val_acc
uracy: 0.7973 - val_loss: 0.5379
Epoch 9/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7875 - loss: 0.5778 - val_acc
uracy: 0.7982 - val_loss: 0.5315
Epoch 10/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7869 - loss: 0.5699 - val_acc
uracy: 0.8008 - val_loss: 0.5290
Epoch 11/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7833 - loss: 0.5706 - val_acc
uracy: 0.7978 - val_loss: 0.5213
Epoch 12/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7940 - loss: 0.5431 - val_acc
uracy: 0.8026 - val_loss: 0.5258
Epoch 13/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8000 - loss: 0.5378 - val_acc
uracy: 0.8100 - val_loss: 0.5122
Epoch 14/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8004 - loss: 0.5311 - val_acc
uracy: 0.8068 - val_loss: 0.5105
Epoch 15/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8043 - loss: 0.5207 - val_acc
uracy: 0.8144 - val_loss: 0.5000
Epoch 16/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8109 - loss: 0.5031 - val_acc
uracy: 0.8074 - val_loss: 0.5080
Epoch 17/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8120 - loss: 0.5000 - val_acc
uracy: 0.8087 - val_loss: 0.5095
Epoch 18/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8117 - loss: 0.5011 - val_acc
uracy: 0.8085 - val_loss: 0.5102
Epoch 19/100
```

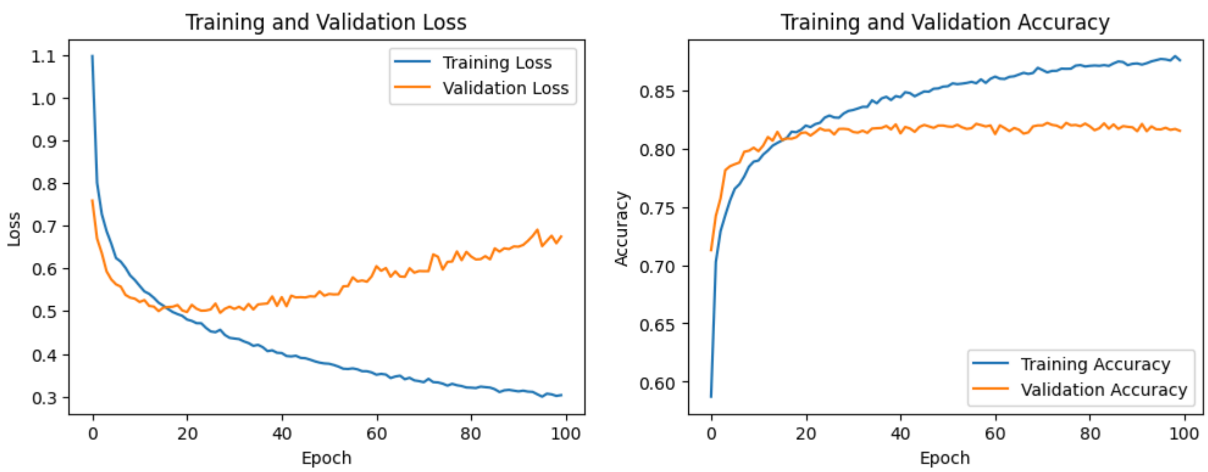
```
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8132 - loss: 0.4906 - val_acc
uracy: 0.8098 - val_loss: 0.5142
Epoch 20/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8194 - loss: 0.4881 - val_acc
uracy: 0.8134 - val_loss: 0.5014
Epoch 21/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8226 - loss: 0.4743 - val_acc
uracy: 0.8139 - val_loss: 0.4979
Epoch 22/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8205 - loss: 0.4744 - val_acc
uracy: 0.8112 - val_loss: 0.5147
Epoch 23/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8246 - loss: 0.4650 - val_acc
uracy: 0.8141 - val_loss: 0.5057
Epoch 24/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8225 - loss: 0.4699 - val_acc
uracy: 0.8174 - val_loss: 0.5009
Epoch 25/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8260 - loss: 0.4595 - val_acc
uracy: 0.8157 - val_loss: 0.5014
Epoch 26/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8329 - loss: 0.4449 - val_acc
uracy: 0.8159 - val_loss: 0.5040
Epoch 27/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8302 - loss: 0.4387 - val_acc
uracy: 0.8122 - val_loss: 0.5176
Epoch 28/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8276 - loss: 0.4473 - val_acc
uracy: 0.8169 - val_loss: 0.4960
Epoch 29/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8279 - loss: 0.4479 - val_acc
uracy: 0.8170 - val_loss: 0.5055
Epoch 30/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8355 - loss: 0.4271 - val_acc
uracy: 0.8166 - val_loss: 0.5103
Epoch 31/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8360 - loss: 0.4325 - val_acc
uracy: 0.8143 - val_loss: 0.5050
Epoch 32/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8341 - loss: 0.4338 - val_acc
uracy: 0.8137 - val_loss: 0.5102
Epoch 33/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8399 - loss: 0.4197 - val_acc
uracy: 0.8154 - val_loss: 0.5031
Epoch 34/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8418 - loss: 0.4124 - val_acc
uracy: 0.8135 - val_loss: 0.5167
Epoch 35/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8482 - loss: 0.4066 - val_acc
uracy: 0.8172 - val_loss: 0.5039
Epoch 36/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8411 - loss: 0.4163 - val_acc
uracy: 0.8175 - val_loss: 0.5156
Epoch 37/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8431 - loss: 0.4124 - val_acc
uracy: 0.8177 - val_loss: 0.5168
```

```
Epoch 38/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8442 - loss: 0.4023 - val_acc
uracy: 0.8195 - val_loss: 0.5179
Epoch 39/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8435 - loss: 0.3991 - val_acc
uracy: 0.8165 - val_loss: 0.5344
Epoch 40/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8482 - loss: 0.3997 - val_acc
uracy: 0.8208 - val_loss: 0.5123
Epoch 41/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8442 - loss: 0.4030 - val_acc
uracy: 0.8131 - val_loss: 0.5328
Epoch 42/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8488 - loss: 0.3969 - val_acc
uracy: 0.8186 - val_loss: 0.5114
Epoch 43/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8475 - loss: 0.3925 - val_acc
uracy: 0.8173 - val_loss: 0.5361
Epoch 44/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8441 - loss: 0.3970 - val_acc
uracy: 0.8144 - val_loss: 0.5321
Epoch 45/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8447 - loss: 0.3904 - val_acc
uracy: 0.8186 - val_loss: 0.5327
Epoch 46/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8534 - loss: 0.3803 - val_acc
uracy: 0.8203 - val_loss: 0.5320
Epoch 47/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8516 - loss: 0.3845 - val_acc
uracy: 0.8191 - val_loss: 0.5347
Epoch 48/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8531 - loss: 0.3784 - val_acc
uracy: 0.8178 - val_loss: 0.5339
Epoch 49/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8565 - loss: 0.3749 - val_acc
uracy: 0.8198 - val_loss: 0.5461
Epoch 50/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8560 - loss: 0.3693 - val_acc
uracy: 0.8198 - val_loss: 0.5360
Epoch 51/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8545 - loss: 0.3732 - val_acc
uracy: 0.8190 - val_loss: 0.5402
Epoch 52/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8597 - loss: 0.3703 - val_acc
uracy: 0.8187 - val_loss: 0.5391
Epoch 53/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8586 - loss: 0.3651 - val_acc
uracy: 0.8205 - val_loss: 0.5395
Epoch 54/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8568 - loss: 0.3554 - val_acc
uracy: 0.8183 - val_loss: 0.5581
Epoch 55/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8605 - loss: 0.3545 - val_acc
uracy: 0.8169 - val_loss: 0.5586
Epoch 56/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8584 - loss: 0.3687 - val_acc
```

```
uracy: 0.8175 - val_loss: 0.5790
Epoch 57/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8573 - loss: 0.3659 - val_acc
uracy: 0.8213 - val_loss: 0.5694
Epoch 58/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8578 - loss: 0.3568 - val_acc
uracy: 0.8202 - val_loss: 0.5717
Epoch 59/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8591 - loss: 0.3546 - val_acc
uracy: 0.8190 - val_loss: 0.5688
Epoch 60/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8597 - loss: 0.3610 - val_acc
uracy: 0.8199 - val_loss: 0.5813
Epoch 61/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8621 - loss: 0.3452 - val_acc
uracy: 0.8125 - val_loss: 0.6050
Epoch 62/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8582 - loss: 0.3473 - val_acc
uracy: 0.8200 - val_loss: 0.5944
Epoch 63/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8586 - loss: 0.3482 - val_acc
uracy: 0.8176 - val_loss: 0.6008
Epoch 64/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8596 - loss: 0.3463 - val_acc
uracy: 0.8150 - val_loss: 0.5805
Epoch 65/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8630 - loss: 0.3486 - val_acc
uracy: 0.8177 - val_loss: 0.5931
Epoch 66/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8652 - loss: 0.3439 - val_acc
uracy: 0.8163 - val_loss: 0.5810
Epoch 67/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8674 - loss: 0.3375 - val_acc
uracy: 0.8129 - val_loss: 0.5802
Epoch 68/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8642 - loss: 0.3408 - val_acc
uracy: 0.8139 - val_loss: 0.6003
Epoch 69/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8668 - loss: 0.3356 - val_acc
uracy: 0.8191 - val_loss: 0.5899
Epoch 70/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8688 - loss: 0.3404 - val_acc
uracy: 0.8199 - val_loss: 0.5938
Epoch 71/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8720 - loss: 0.3263 - val_acc
uracy: 0.8199 - val_loss: 0.5935
Epoch 72/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8701 - loss: 0.3288 - val_acc
uracy: 0.8221 - val_loss: 0.5935
Epoch 73/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8679 - loss: 0.3320 - val_acc
uracy: 0.8204 - val_loss: 0.6328
Epoch 74/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8667 - loss: 0.3316 - val_acc
uracy: 0.8197 - val_loss: 0.6271
Epoch 75/100
```

```
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8694 - loss: 0.3234 - val_acc
uracy: 0.8177 - val_loss: 0.5973
Epoch 76/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8700 - loss: 0.3222 - val_acc
uracy: 0.8221 - val_loss: 0.6151
Epoch 77/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8708 - loss: 0.3226 - val_acc
uracy: 0.8200 - val_loss: 0.6161
Epoch 78/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8765 - loss: 0.3177 - val_acc
uracy: 0.8203 - val_loss: 0.6396
Epoch 79/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8742 - loss: 0.3159 - val_acc
uracy: 0.8192 - val_loss: 0.6192
Epoch 80/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8743 - loss: 0.3165 - val_acc
uracy: 0.8215 - val_loss: 0.6388
Epoch 81/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8727 - loss: 0.3179 - val_acc
uracy: 0.8191 - val_loss: 0.6277
Epoch 82/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8708 - loss: 0.3194 - val_acc
uracy: 0.8158 - val_loss: 0.6209
Epoch 83/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8727 - loss: 0.3173 - val_acc
uracy: 0.8175 - val_loss: 0.6216
Epoch 84/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8729 - loss: 0.3148 - val_acc
uracy: 0.8217 - val_loss: 0.6286
Epoch 85/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8726 - loss: 0.3126 - val_acc
uracy: 0.8171 - val_loss: 0.6211
Epoch 86/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8739 - loss: 0.3113 - val_acc
uracy: 0.8208 - val_loss: 0.6469
Epoch 87/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8749 - loss: 0.3077 - val_acc
uracy: 0.8169 - val_loss: 0.6392
Epoch 88/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8762 - loss: 0.3124 - val_acc
uracy: 0.8193 - val_loss: 0.6469
Epoch 89/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8754 - loss: 0.3050 - val_acc
uracy: 0.8187 - val_loss: 0.6451
Epoch 90/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8764 - loss: 0.3104 - val_acc
uracy: 0.8183 - val_loss: 0.6513
Epoch 91/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8763 - loss: 0.2986 - val_acc
uracy: 0.8149 - val_loss: 0.6507
Epoch 92/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8737 - loss: 0.3157 - val_acc
uracy: 0.8212 - val_loss: 0.6549
Epoch 93/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8761 - loss: 0.3067 - val_acc
uracy: 0.8149 - val_loss: 0.6645
```

```
Epoch 94/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8739 - loss: 0.3118 - val_acc
uracy: 0.8191 - val_loss: 0.6760
Epoch 95/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8764 - loss: 0.3038 - val_acc
uracy: 0.8166 - val_loss: 0.6906
Epoch 96/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8759 - loss: 0.2945 - val_acc
uracy: 0.8165 - val_loss: 0.6521
Epoch 97/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8779 - loss: 0.3068 - val_acc
uracy: 0.8179 - val_loss: 0.6644
Epoch 98/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8769 - loss: 0.3038 - val_acc
uracy: 0.8161 - val_loss: 0.6765
Epoch 99/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8828 - loss: 0.2897 - val_acc
uracy: 0.8169 - val_loss: 0.6588
Epoch 100/100
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8777 - loss: 0.2985 - val_acc
uracy: 0.8153 - val_loss: 0.6744
Final Training Accuracy: 0.8758999705314636
Final Validation Accuracy: 0.8152999877929688
```



8: Observations

As we can see, there are some impactfull changes with adjusting the convolution layers in our model.

1. At the 1st version of our CNN model with 2 convolution layers, 100 fully connected neurons and Dropout = 0.3,the validation loss is around 0.88 and the validation accuracyis around 0.82
2. At the 2nd version of our CNN model with 3 convolution layers, 50 fully connected neurons and Dropout = 0.15, the validation loss is around 1.5 and the validation accuracy is around 0.82
3. At the 1st version of our CNN model with 2 convolution layers, the validation loss is around 0.68 and the validation accuracy is around 0.82

We can conclude that the best model of the 3 versions for the CNN is the last one because:

- It has the minimum validation loss score on unseen data
- It has the same validation accuracy score as the other versions

Reasons that might cause the validation loss score to be greater than 1:

- Overfitting
- High Learn Rate

FOR PDF Extraction:

- Install pandoc from https://github.com/jgm/pandoc/releases/tag/3.1.13
- pip install nbconvert[webpdf]

Open Terminal

- Navigate to the directory that contains the file ml_1.ipynb
- Enter: **jupyter nbconvert --to webpdf --allow-chromium-download ml_1.ipynb** at the command prompt