המכללה
האקדמית להנדסה
בראודה
בכרמיאל

המחלקה להנדסת חשמל ואלקטרוניקה

(31245) מערכות לומדות ולמידה עמוקה

Lab 10 report

פרנסיס עבוד

Prepared for: Dr. Amer Adler

Date: 27/06/2025

Deep Learning – Lab – Sentiment Analysis

1. Load the MLP-based and 1D-CNN-based IMDB Sentiment Analysis solutions into Colab.

2. Train and compute the test data accuracy. Plot the loss and accuracy training curves.

3. Add early stopping to find the optimal stopping point. Is there any improvement compared to the baseline accuracy?

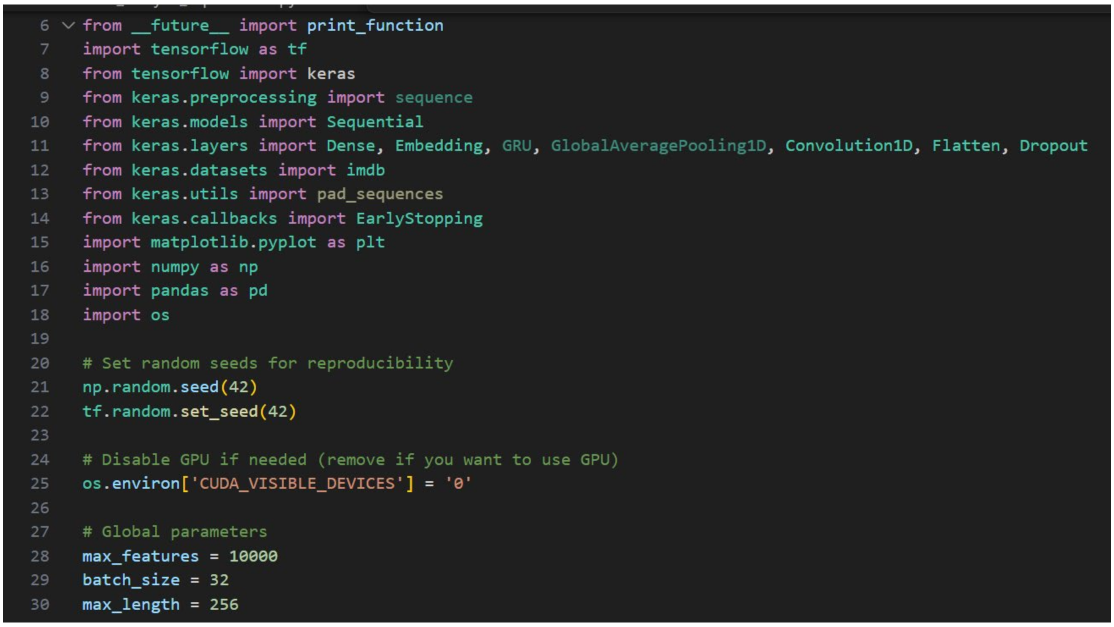
4. In the CNN network, change the embedding layer dimension to a few different values, re-train and compute the test data accuracy. Which value provides the best accuracy?

5. Repeat step 4, for the MLP network.

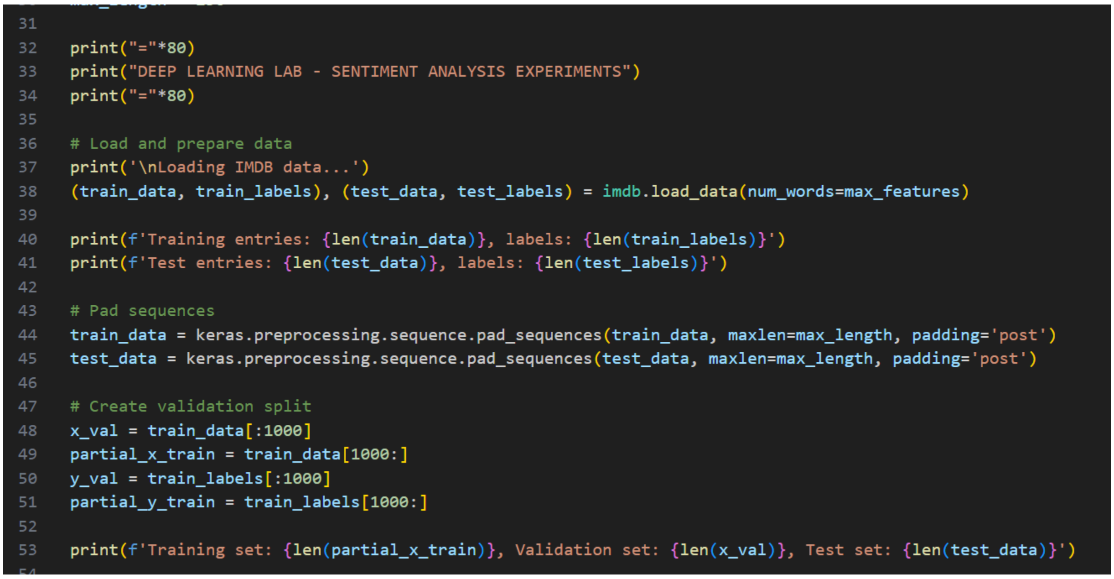
6. In the CNN network, remove 1 convolutional layer, and compute accuracy, afterwards remove 2 layers. Try also to change the kernel length. What is the impact on the accuracy?

7. In the MLP network, increase the FC layer to a higher number of neurons, and compute accuracy. Which value provides the best accuracy?

Solution:



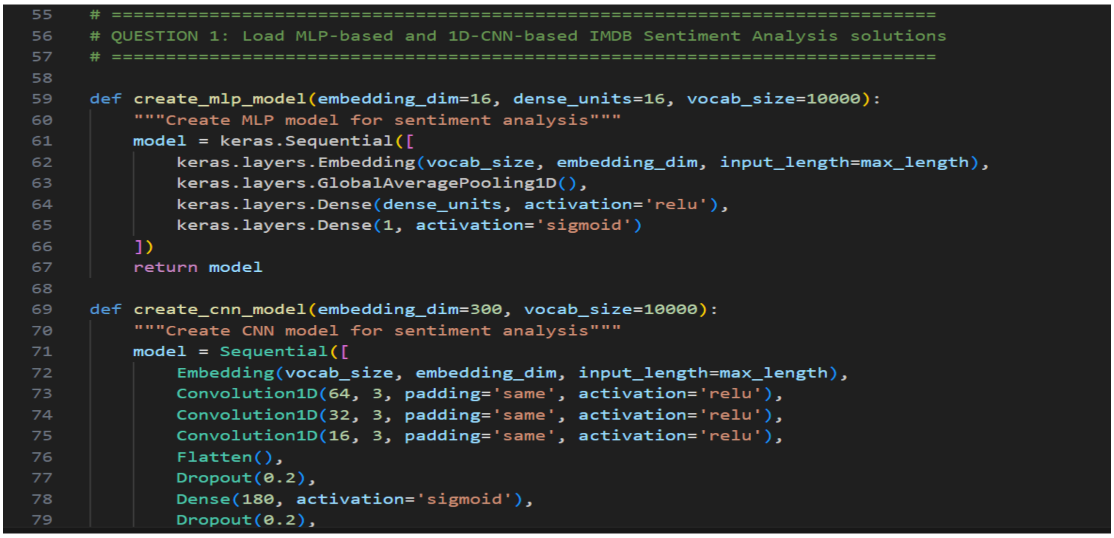
```python
from __future__ import print_function
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding, GRU, GlobalAveragePooling1D, Convolution1D, Flatten, Dropout
from keras.datasets import imdb
from keras.utils import pad_sequences
from keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Disable GPU if needed (remove if you want to use GPU)
os.environ['CUDA_VISIBLE_DEVICES'] = '0'

# Global parameters
max_features = 10000
batch_size = 32
max_length = 256
```



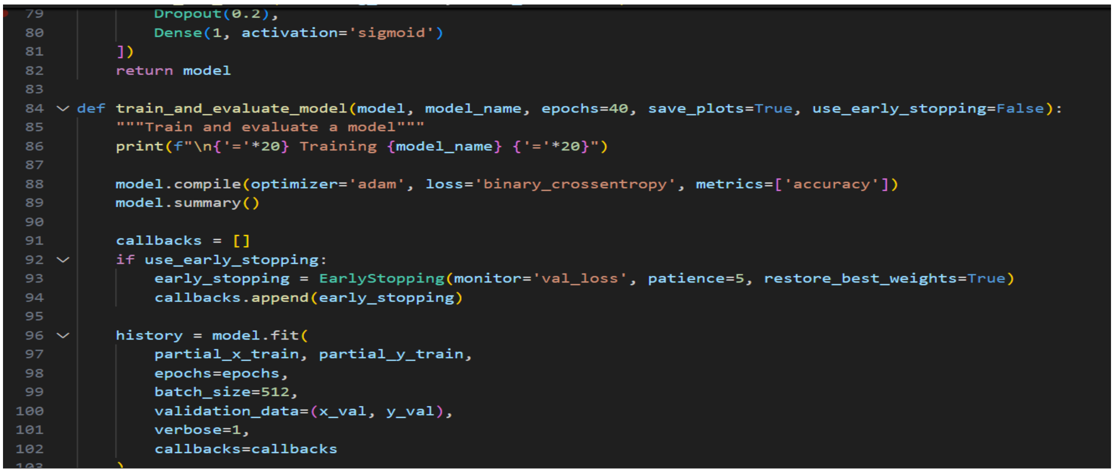
```python

print("="*80)
print("DEEP LEARNING LAB - SENTIMENT ANALYSIS EXPERIMENTS")
print("="*80)

# Load and prepare data
print('\nLoading IMDB data...')
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=max_features)

print(f'Training entries: {len(train_data)}, labels: {len(train_labels)}')
print(f'Test entries: {len(test_data)}, labels: {len(test_labels)}')

# Pad sequences
train_data = keras.preprocessing.sequence.pad_sequences(train_data, maxlen=max_length, padding='post')
test_data = keras.preprocessing.sequence.pad_sequences(test_data, maxlen=max_length, padding='post')

# Create validation split
x_val = train_data[:1000]
partial_x_train = train_data[1000:]
y_val = train_labels[:1000]
partial_y_train = train_labels[1000:]

print(f'Training set: {len(partial_x_train)}, Validation set: {len(x_val)}, Test set: {len(test_data)}')
```

1. Load the MLP-based and 1D-CNN-based IMDB Sentiment Analysis solutions into Colab:

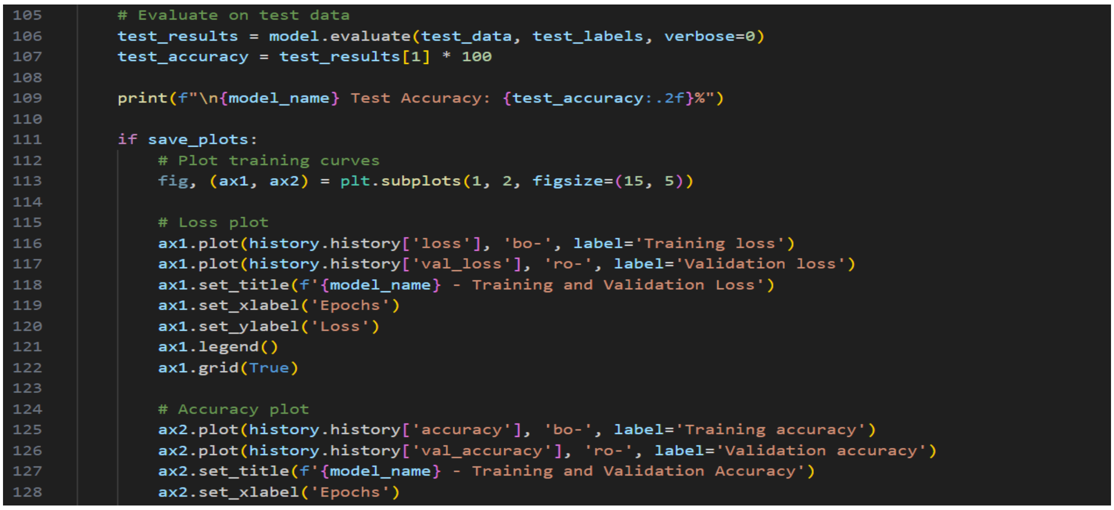
Code:



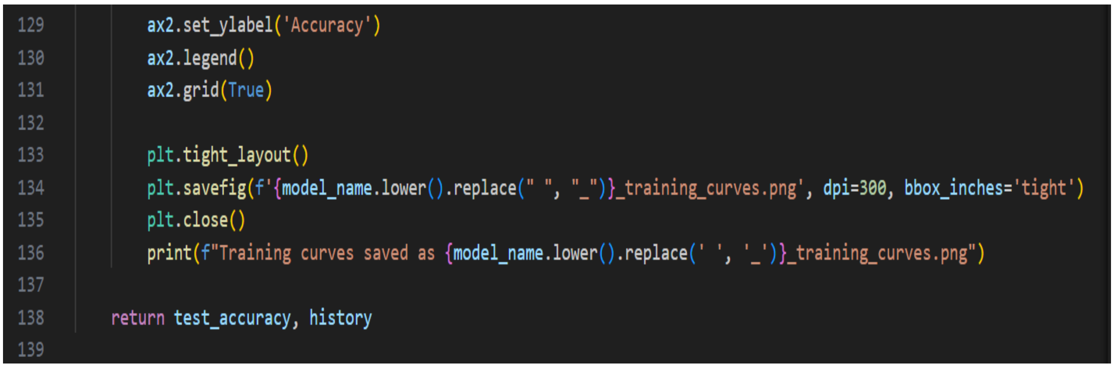
```python
# ============================================================================
# QUESTION 1: Load MLP-based and 1D-CNN-based IMDB Sentiment Analysis solutions
# ============================================================================

def create_mlp_model(embedding_dim=16, dense_units=16, vocab_size=10000):
    """Create MLP model for sentiment analysis"""
    model = keras.Sequential([
        keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
        keras.layers.GlobalAveragePooling1D(),
        keras.layers.Dense(dense_units, activation='relu'),
        keras.layers.Dense(1, activation='sigmoid')
    ])
    return model

def create_cnn_model(embedding_dim=300, vocab_size=10000):
    """Create CNN model for sentiment analysis"""
    model = Sequential([
        Embedding(vocab_size, embedding_dim, input_length=max_length),
        Convolution1D(64, 3, padding='same', activation='relu'),
        Convolution1D(32, 3, padding='same', activation='relu'),
        Convolution1D(16, 3, padding='same', activation='relu'),
        Flatten(),
        Dropout(0.2),
        Dense(180, activation='sigmoid'),
        Dropout(0.2),
```



```python
        Dropout(0.2),
        Dense(1, activation='sigmoid')
    ])
    return model

def train_and_evaluate_model(model, model_name, epochs=40, save_plots=True, use_early_stopping=False):
    """Train and evaluate a model"""
    print(f"\n{'='*20} Training {model_name} {'='*20}")

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    model.summary()

    callbacks = []
    if use_early_stopping:
        early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
        callbacks.append(early_stopping)

    history = model.fit(
        partial_x_train, partial_y_train,
        epochs=epochs,
        batch_size=512,
        validation_data=(x_val, y_val),
        verbose=1,
        callbacks=callbacks
    )
```



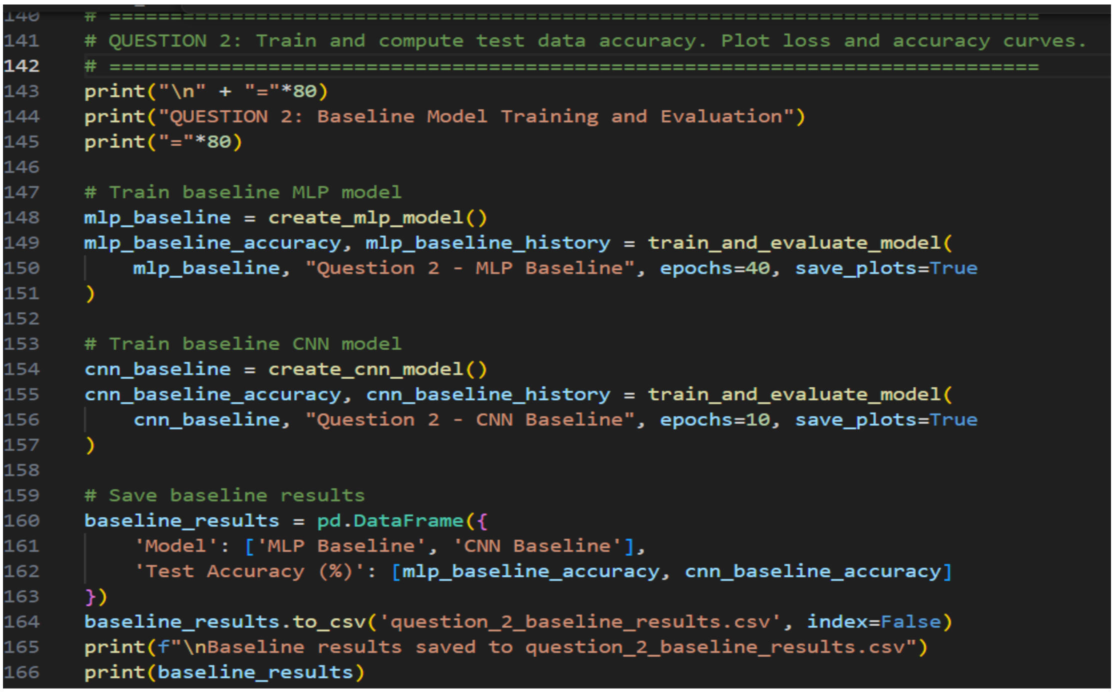
```
    # Evaluate on test data
    test_results = model.evaluate(test_data, test_labels, verbose=0)
    test_accuracy = test_results[1] * 100

    print(f"\n{model_name} Test Accuracy: {test_accuracy:.2f}%")

    if save_plots:
        # Plot training curves
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

        # Loss plot
        ax1.plot(history.history['loss'], 'bo-', label='Training loss')
        ax1.plot(history.history['val_loss'], 'ro-', label='Validation loss')
        ax1.set_title(f'{model_name} - Training and Validation Loss')
        ax1.set_xlabel('Epochs')
        ax1.set_ylabel('Loss')
        ax1.legend()
        ax1.grid(True)

        # Accuracy plot
        ax2.plot(history.history['accuracy'], 'bo-', label='Training accuracy')
        ax2.plot(history.history['val_accuracy'], 'ro-', label='Validation accuracy')
        ax2.set_title(f'{model_name} - Training and Validation Accuracy')
        ax2.set_xlabel('Epochs')
```



```
        ax2.set_ylabel('Accuracy')
        ax2.legend()
        ax2.grid(True)

        plt.tight_layout()
        plt.savefig(f'{model_name.lower().replace(" ", "_")}_training_curves.png', dpi=300, bbox_inches='tight')
        plt.close()
        print(f"Training curves saved as {model_name.lower().replace(' ', '_')}_training_curves.png")

    return test_accuracy, history

```

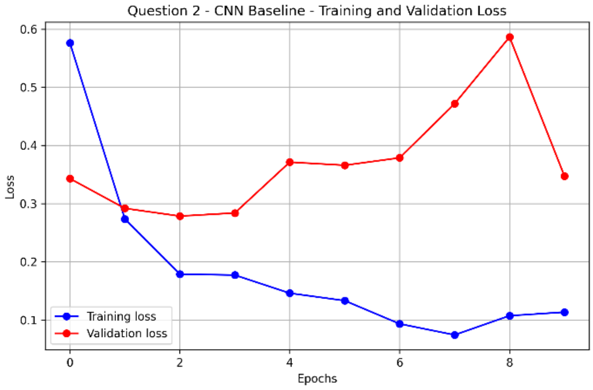
2. Train and compute the test data accuracy. Plot the loss and accuracy training curves.

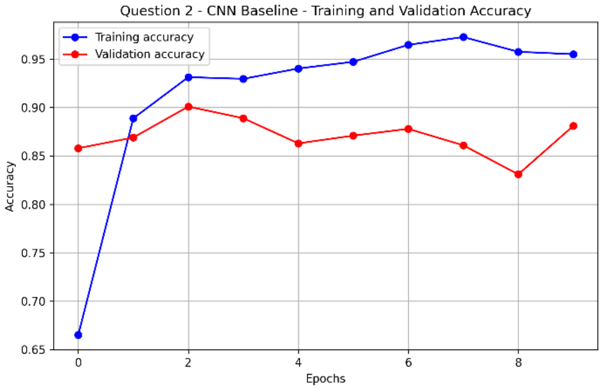
Code:

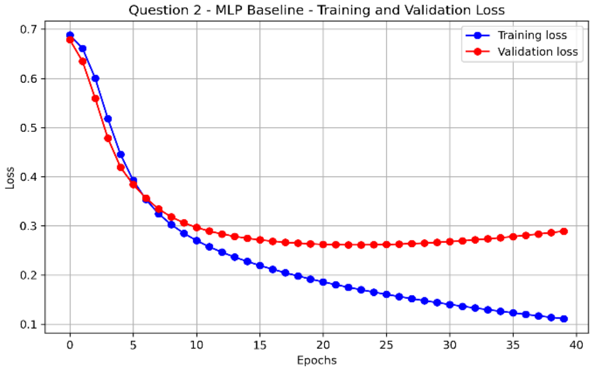


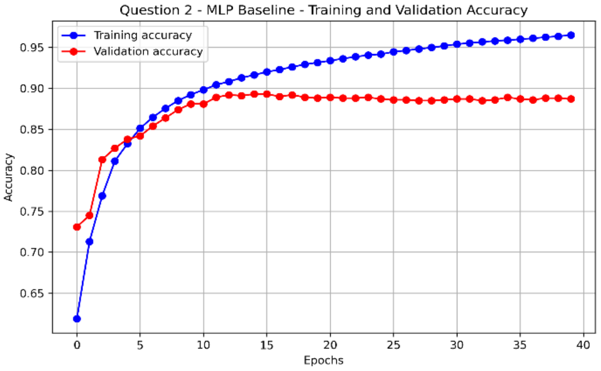
```
# ============================================================================
# QUESTION 2: Train and compute test data accuracy. Plot loss and accuracy curves.
# ============================================================================
print("\n" + "="*80)
print("QUESTION 2: Baseline Model Training and Evaluation")
print("="*80)

# Train baseline MLP model
mlp_baseline = create_mlp_model()
mlp_baseline_accuracy, mlp_baseline_history = train_and_evaluate_model(
    mlp_baseline, "Question 2 - MLP Baseline", epochs=40, save_plots=True
)

# Train baseline CNN model
cnn_baseline = create_cnn_model()
cnn_baseline_accuracy, cnn_baseline_history = train_and_evaluate_model(
    cnn_baseline, "Question 2 - CNN Baseline", epochs=10, save_plots=True
)

# Save baseline results
baseline_results = pd.DataFrame({
    'Model': ['MLP Baseline', 'CNN Baseline'],
    'Test Accuracy (%)': [mlp_baseline_accuracy, cnn_baseline_accuracy]
})
baseline_results.to_csv('question_2_baseline_results.csv', index=False)
print(f"\nBaseline results saved to question_2_baseline_results.csv")
print(baseline_results)
```

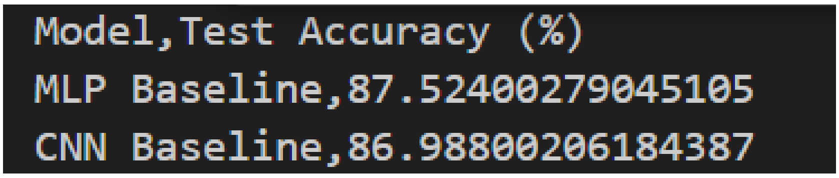
Output for Q-2:







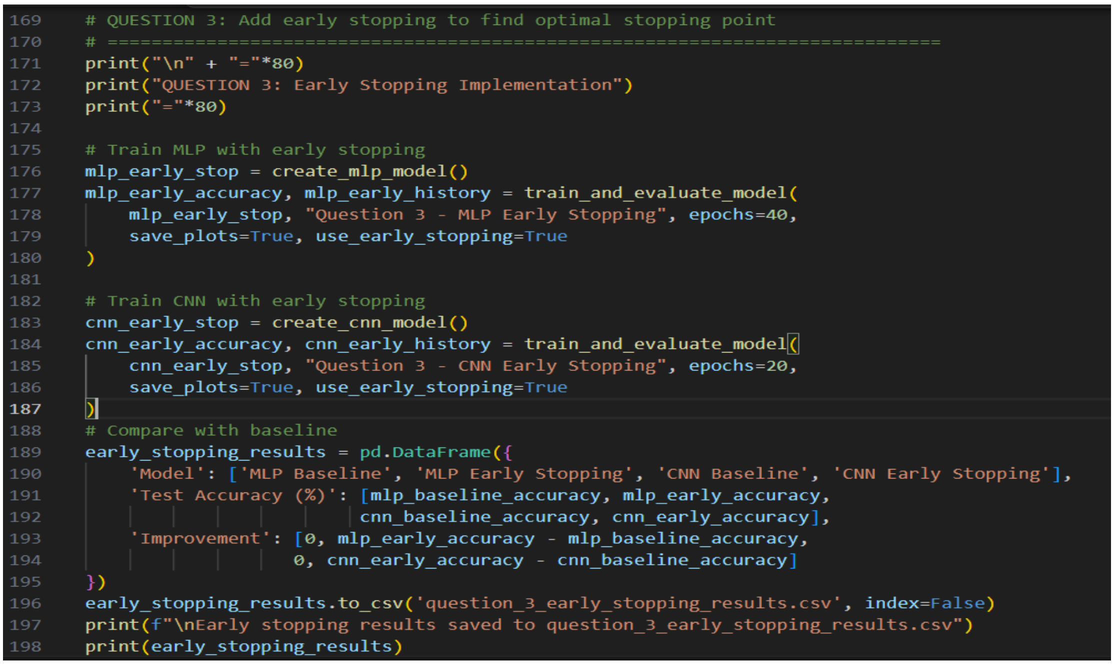




```
Model,Test Accuracy (%)
MLP Baseline,87.52400279045105
CNN Baseline,86.98800206184387
```

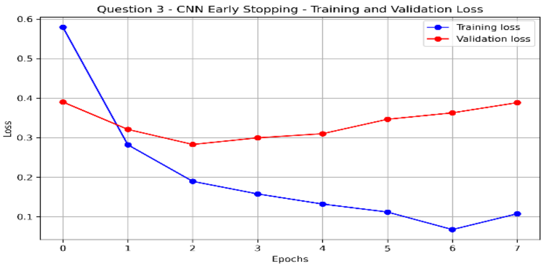
3. Add early stopping to find the optimal stopping point. Is there any improvement compared to the baseline accuracy?

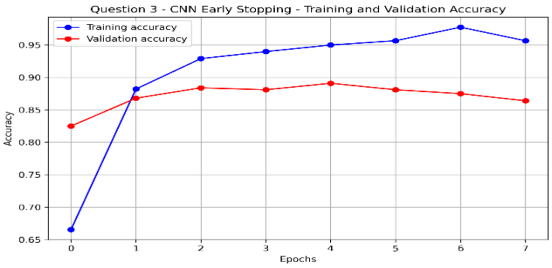
Code:

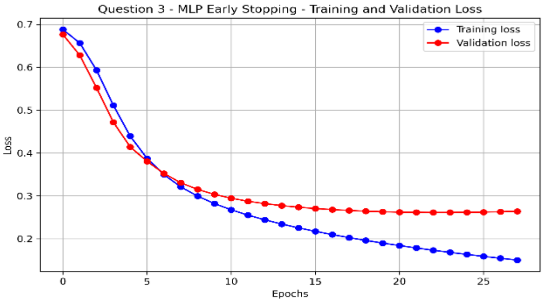


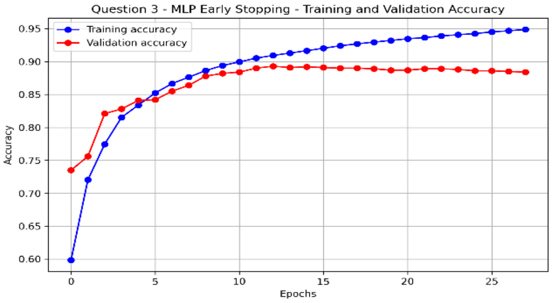
```
# QUESTION 3: Add early stopping to find optimal stopping point
# ==========================================================================
print("\n" + "="*80)
print("QUESTION 3: Early Stopping Implementation")
print("="*80)

# Train MLP with early stopping
mlp_early_stop = create_mlp_model()
mlp_early_accuracy, mlp_early_history = train_and_evaluate_model(
    mlp_early_stop, "Question 3 - MLP Early Stopping", epochs=40,
    save_plots=True, use_early_stopping=True
)

# Train CNN with early stopping
cnn_early_stop = create_cnn_model()
cnn_early_accuracy, cnn_early_history = train_and_evaluate_model(
    cnn_early_stop, "Question 3 - CNN Early Stopping", epochs=20,
    save_plots=True, use_early_stopping=True
)
# Compare with baseline
early_stopping_results = pd.DataFrame({
    'Model': ['MLP Baseline', 'MLP Early Stopping', 'CNN Baseline', 'CNN Early Stopping'],
    'Test Accuracy (%)': [mlp_baseline_accuracy, mlp_early_accuracy,
                          cnn_baseline_accuracy, cnn_early_accuracy],
    'Improvement': [0, mlp_early_accuracy - mlp_baseline_accuracy,
                   0, cnn_early_accuracy - cnn_baseline_accuracy]
})
early_stopping_results.to_csv('question_3_early_stopping_results.csv', index=False)
print(f"\nEarly stopping results saved to question_3_early_stopping_results.csv")
print(early_stopping_results)
```

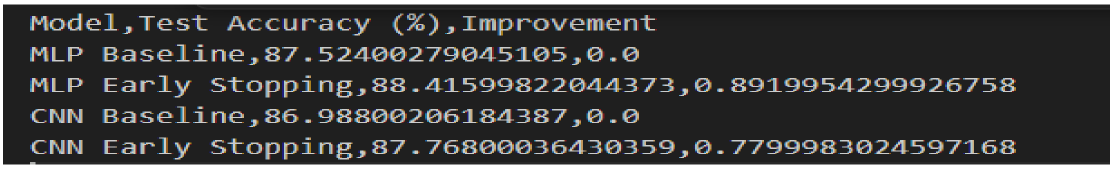
Output for Q-3:







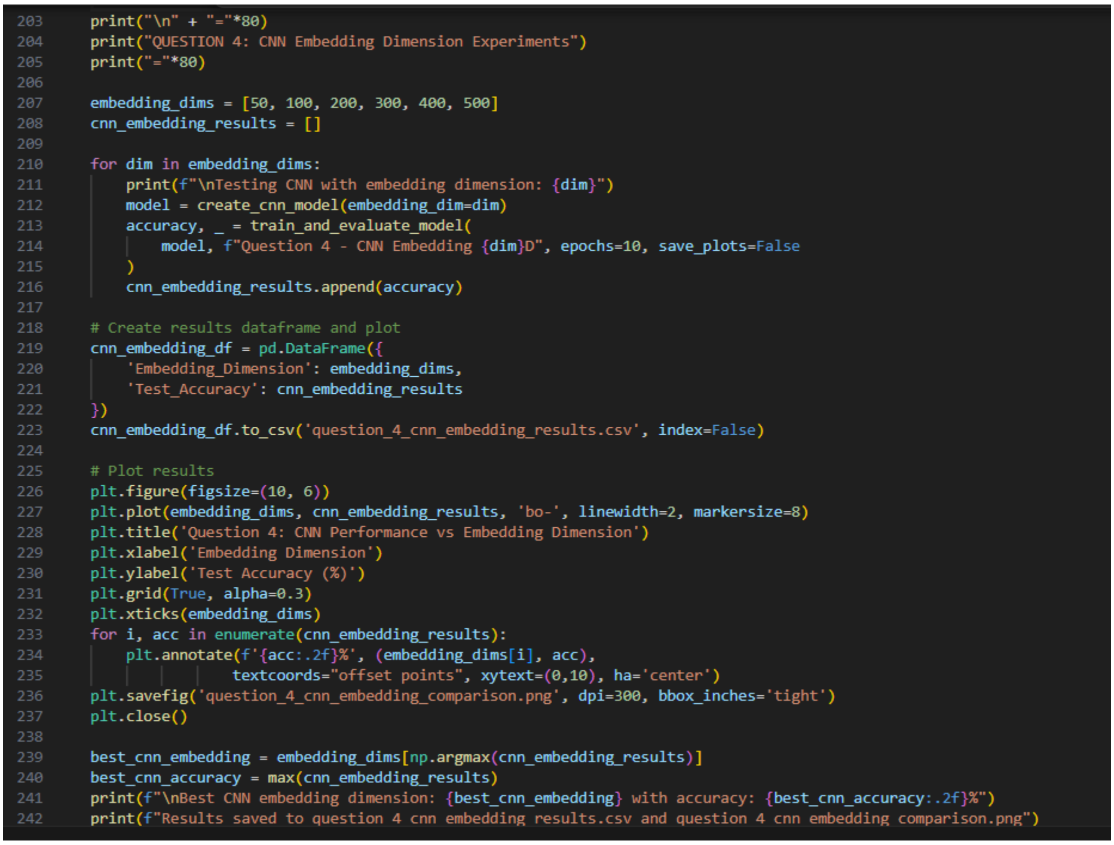




```
Model,Test Accuracy (%),Improvement
MLP Baseline,87.52400279045105,0.0
MLP Early Stopping,88.41599822044373,0.8919954299926758
CNN Baseline,86.98800206184387,0.0
CNN Early Stopping,87.76800036430359,0.7799983024597168
```

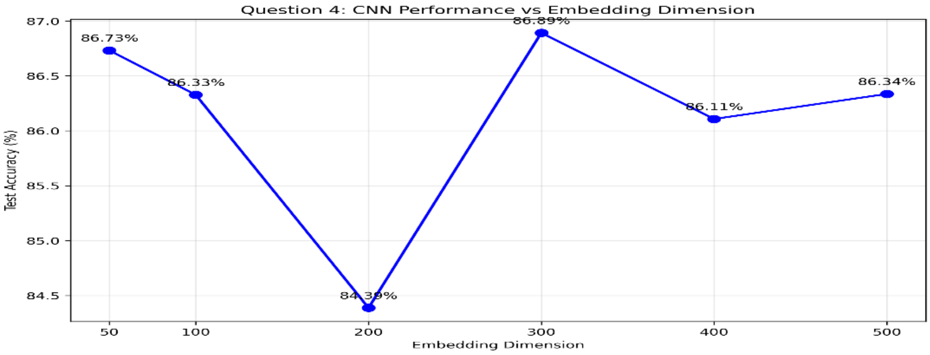
4. In the CNN network, change the embedding layer dimension to a few different values, re-train and compute the test data accuracy. Which value provides the best accuracy?

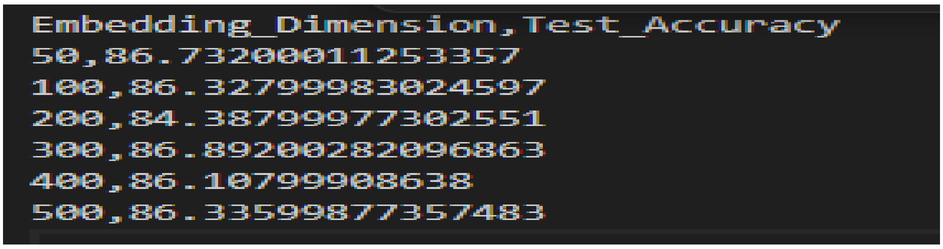
Code:



```python
print("\n" + "="*80)
print("QUESTION 4: CNN Embedding Dimension Experiments")
print("="*80)

embedding_dims = [50, 100, 200, 300, 400, 500]
cnn_embedding_results = []

for dim in embedding_dims:
    print(f"\nTesting CNN with embedding dimension: {dim}")
    model = create_cnn_model(embedding_dim=dim)
    accuracy, _ = train_and_evaluate_model(
        model, f"Question 4 - CNN Embedding {dim}D", epochs=10, save_plots=False
    )
    cnn_embedding_results.append(accuracy)

# Create results dataframe and plot
cnn_embedding_df = pd.DataFrame({
    'Embedding_Dimension': embedding_dims,
    'Test_Accuracy': cnn_embedding_results
})
cnn_embedding_df.to_csv('question_4_cnn_embedding_results.csv', index=False)

# Plot results
plt.figure(figsize=(10, 6))
plt.plot(embedding_dims, cnn_embedding_results, 'bo-', linewidth=2, markersize=8)
plt.title('Question 4: CNN Performance vs Embedding Dimension')
plt.xlabel('Embedding Dimension')
plt.ylabel('Test Accuracy (%)')
plt.grid(True, alpha=0.3)
plt.xticks(embedding_dims)
for i, acc in enumerate(cnn_embedding_results):
    plt.annotate(f'{acc:.2f}%', (embedding_dims[i], acc),
                 textcoords="offset points", xytext=(0,10), ha='center')
plt.savefig('question_4_cnn_embedding_comparison.png', dpi=300, bbox_inches='tight')
plt.close()

best_cnn_embedding = embedding_dims[np.argmax(cnn_embedding_results)]
best_cnn_accuracy = max(cnn_embedding_results)
print(f"\nBest CNN embedding dimension: {best_cnn_embedding} with accuracy: {best_cnn_accuracy:.2f}%")
print(f"Results saved to question_4_cnn_embedding_results.csv and question_4_cnn_embedding_comparison.png")
```

Output for Q-4:

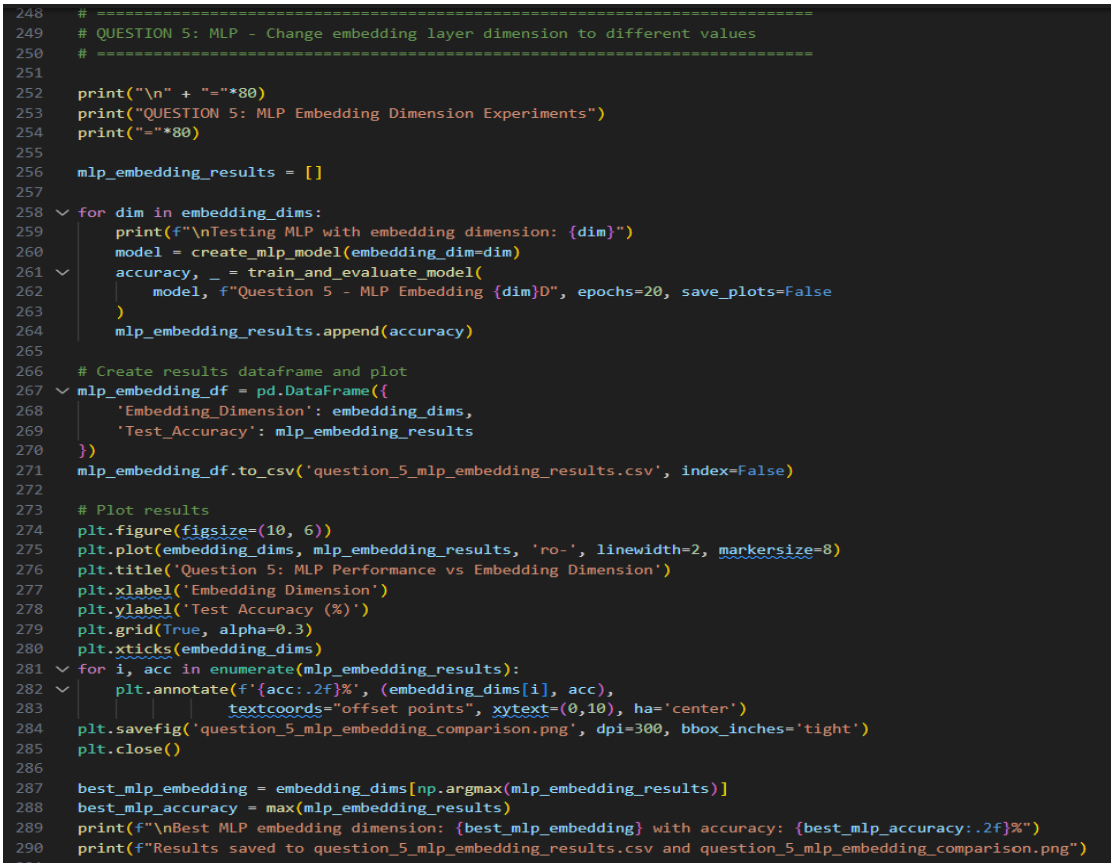




```
Embedding_Dimension,Test_Accuracy
50,86.73200011253357
100,86.32799983024597
200,84.38799977302551
300,86.89200282096863
400,86.10799908638
500,86.33599877357483
```

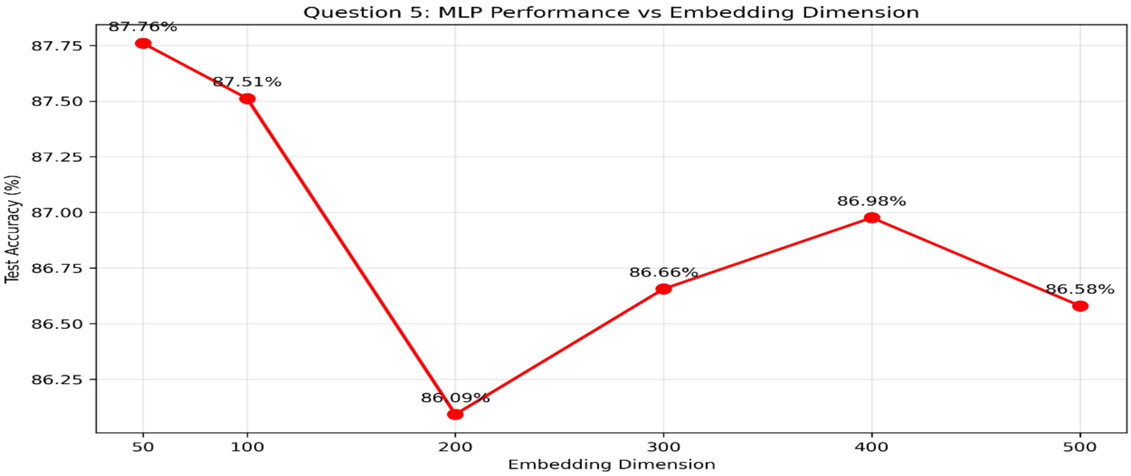
5. Repeat step 4, for the MLP network.

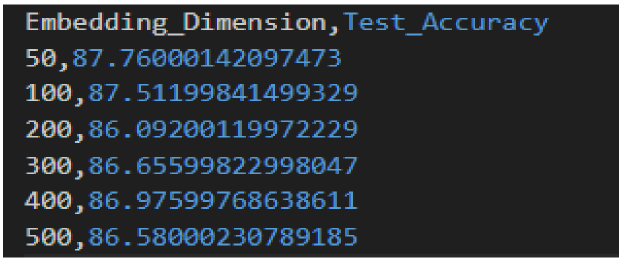
Code:



```
# ============================================================================
# QUESTION 5: MLP - Change embedding layer dimension to different values
# ============================================================================

print("\n" + "="*80)
print("QUESTION 5: MLP Embedding Dimension Experiments")
print("="*80)

mlp_embedding_results = []

for dim in embedding_dims:
    print(f"\nTesting MLP with embedding dimension: {dim}")
    model = create_mlp_model(embedding_dim=dim)
    accuracy, _ = train_and_evaluate_model(
        model, f"Question 5 - MLP Embedding {dim}D", epochs=20, save_plots=False
    )
    mlp_embedding_results.append(accuracy)

# Create results dataframe and plot
mlp_embedding_df = pd.DataFrame({
    'Embedding_Dimension': embedding_dims,
    'Test_Accuracy': mlp_embedding_results
})
mlp_embedding_df.to_csv('question_5_mlp_embedding_results.csv', index=False)

# Plot results
plt.figure(figsize=(10, 6))
plt.plot(embedding_dims, mlp_embedding_results, 'ro-', linewidth=2, markersize=8)
plt.title('Question 5: MLP Performance vs Embedding Dimension')
plt.xlabel('Embedding Dimension')
plt.ylabel('Test Accuracy (%)')
plt.grid(True, alpha=0.3)
plt.xticks(embedding_dims)
for i, acc in enumerate(mlp_embedding_results):
    plt.annotate(f'{acc:.2f}%', (embedding_dims[i], acc),
                        textcoords="offset points", xytext=(0,10), ha='center')
plt.savefig('question_5_mlp_embedding_comparison.png', dpi=300, bbox_inches='tight')
plt.close()

best_mlp_embedding = embedding_dims[np.argmax(mlp_embedding_results)]
best_mlp_accuracy = max(mlp_embedding_results)
print(f"\nBest MLP embedding dimension: {best_mlp_embedding} with accuracy: {best_mlp_accuracy:.2f}%")
print(f"Results saved to question_5_mlp_embedding_results.csv and question_5_mlp_embedding_comparison.png")
```

Output for Q-5:

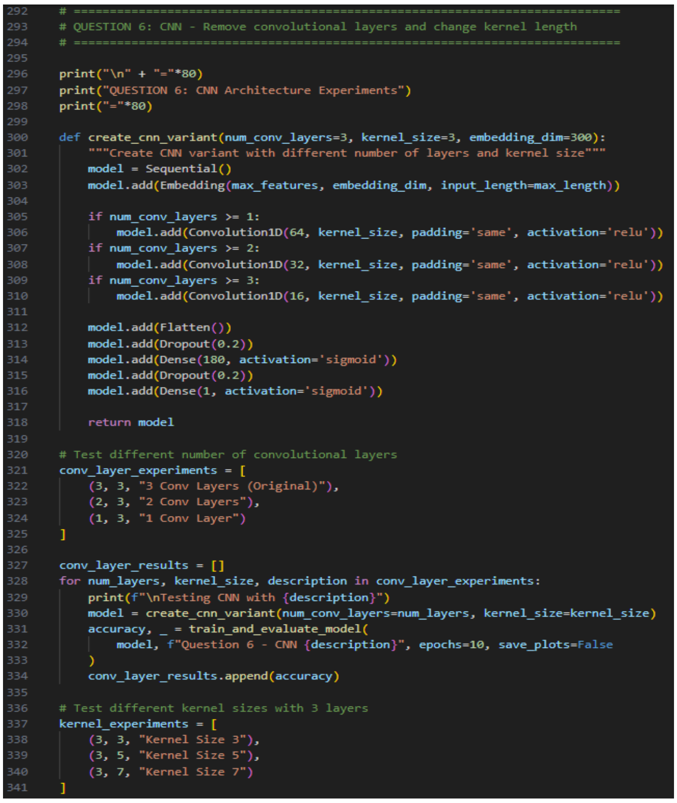




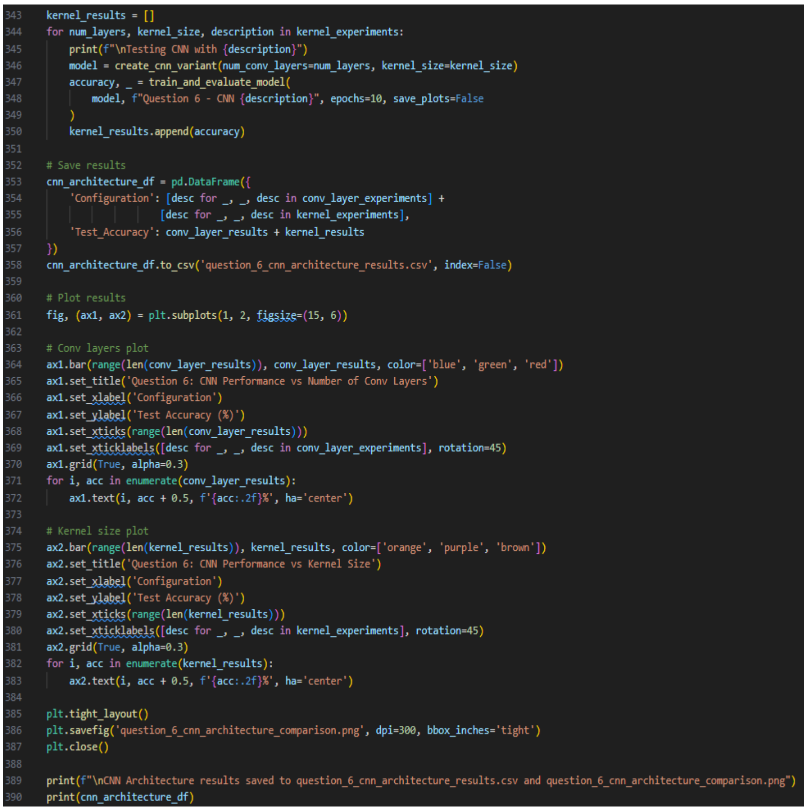
```
Embedding_Dimension,Test_Accuracy
50,87.76000142097473
100,87.51199841499329
200,86.09200119972229
300,86.65599822998047
400,86.97599768638611
500,86.58000230789185
```

6. In the CNN network, remove 1 convolutional layer, and compute accuracy, afterwards remove 2 layers. Try also to change the kernel length. What is the impact on the accuracy?

Code:

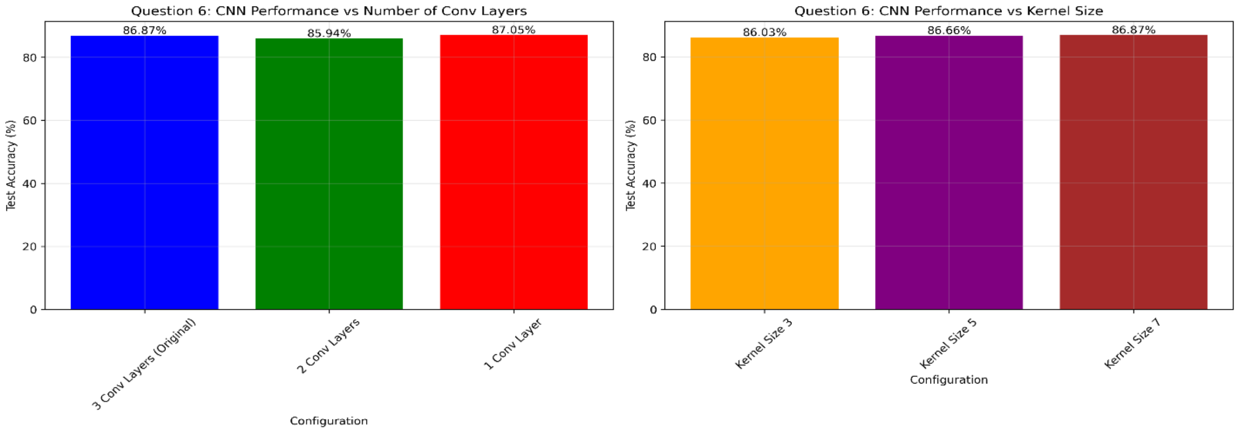


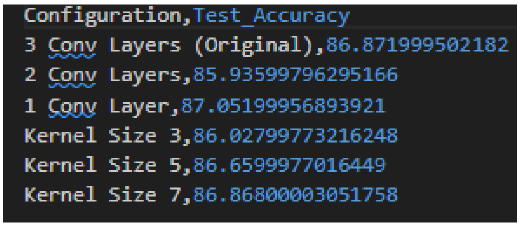
```python
# =============================================================================
# QUESTION 6: CNN - Remove convolutional layers and change kernel length
# =============================================================================

print("\n" + "="*80)
print("QUESTION 6: CNN Architecture Experiments")
print("="*80)

def create_cnn_variant(num_conv_layers=3, kernel_size=3, embedding_dim=300):
    """Create CNN variant with different number of layers and kernel size"""
    model = Sequential()
    model.add(Embedding(max_features, embedding_dim, input_length=max_length))

    if num_conv_layers >= 1:
        model.add(Convolution1D(64, kernel_size, padding='same', activation='relu'))
    if num_conv_layers >= 2:
        model.add(Convolution1D(32, kernel_size, padding='same', activation='relu'))
    if num_conv_layers >= 3:
        model.add(Convolution1D(16, kernel_size, padding='same', activation='relu'))

    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(180, activation='sigmoid'))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))

    return model

# Test different number of convolutional layers
conv_layer_experiments = [
    (3, 3, "3 Conv Layers (Original)"),
    (2, 3, "2 Conv Layers"),
    (1, 3, "1 Conv Layer")
]

conv_layer_results = []
for num_layers, kernel_size, description in conv_layer_experiments:
    print(f"\nTesting CNN with {description}")
    model = create_cnn_variant(num_conv_layers=num_layers, kernel_size=kernel_size)
    accuracy, _ = train_and_evaluate_model(
        model, f"Question 6 - CNN {description}", epochs=10, save_plots=False
    )
    conv_layer_results.append(accuracy)

# Test different kernel sizes with 3 layers
kernel_experiments = [
    (3, 3, "Kernel Size 3"),
    (3, 5, "Kernel Size 5"),
    (3, 7, "Kernel Size 7")
]
```



```python
kernel_results = []
for num_layers, kernel_size, description in kernel_experiments:
    print(f"\nTesting CNN with {description}")
    model = create_cnn_variant(num_conv_layers=num_layers, kernel_size=kernel_size)
    accuracy, _ = train_and_evaluate_model(
        model, f"Question 6 - CNN {description}", epochs=10, save_plots=False
    )
    kernel_results.append(accuracy)

# Save results
cnn_architecture_df = pd.DataFrame({
    'Configuration': [desc for _, _, desc in conv_layer_experiments] +
                     [desc for _, _, desc in kernel_experiments],
    'Test_Accuracy': conv_layer_results + kernel_results
})
cnn_architecture_df.to_csv('question_6_cnn_architecture_results.csv', index=False)

# Plot results
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# Conv layers plot
ax1.bar(range(len(conv_layer_results)), conv_layer_results, color=['blue', 'green', 'red'])
ax1.set_title('Question 6: CNN Performance vs Number of Conv Layers')
ax1.set_xlabel('Configuration')
ax1.set_ylabel('Test Accuracy (%)')
ax1.set_xticks(range(len(conv_layer_results)))
ax1.set_xticklabels([desc for _, _, desc in conv_layer_experiments], rotation=45)
ax1.grid(True, alpha=0.3)
for i, acc in enumerate(conv_layer_results):
    ax1.text(i, acc + 0.5, f'{acc:.2f}%', ha='center')

# Kernel size plot
ax2.bar(range(len(kernel_results)), kernel_results, color=['orange', 'purple', 'brown'])
ax2.set_title('Question 6: CNN Performance vs Kernel Size')
ax2.set_xlabel('Configuration')
ax2.set_ylabel('Test Accuracy (%)')
ax2.set_xticks(range(len(kernel_results)))
ax2.set_xticklabels([desc for _, _, desc in kernel_experiments], rotation=45)
ax2.grid(True, alpha=0.3)
for i, acc in enumerate(kernel_results):
    ax2.text(i, acc + 0.5, f'{acc:.2f}%', ha='center')

plt.tight_layout()
plt.savefig('question_6_cnn_architecture_comparison.png', dpi=300, bbox_inches='tight')
plt.close()

print(f"\nCNN Architecture results saved to question_6_cnn_architecture_results.csv and question_6_cnn_architecture_comparison.png")
print(cnn_architecture_df)
```

Output for Q-6:

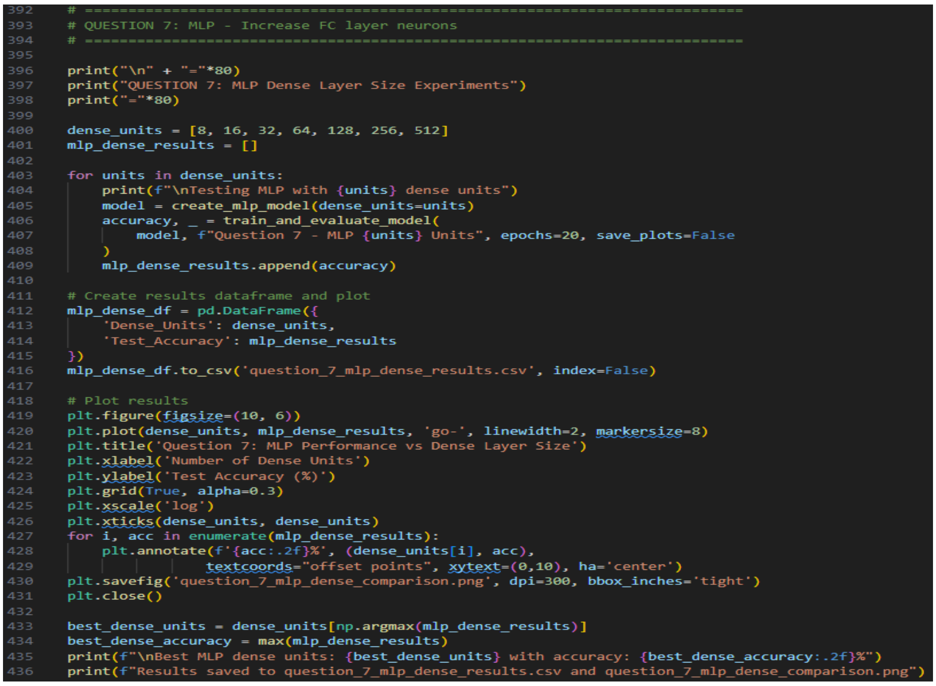




```
Configuration,Test_Accuracy
3 Conv Layers (Original),86.871999502182
2 Conv Layers,85.93599796295166
1 Conv Layer,87.05199956893921
Kernel Size 3,86.02799773216248
Kernel Size 5,86.6599977016449
Kernel Size 7,86.86800003051758
```

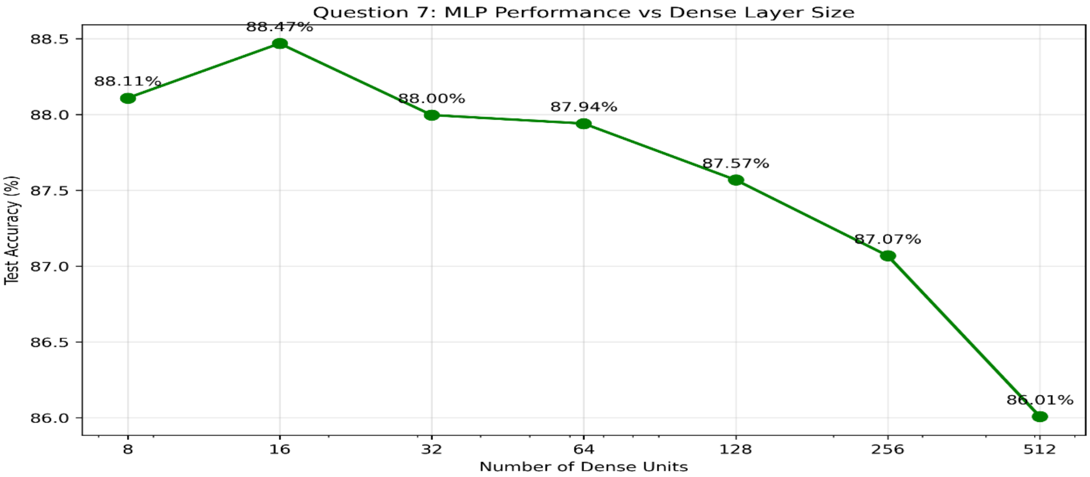
7. In the MLP network, increase the FC layer to a higher number of neurons, and compute accuracy. Which value provides the best accuracy?

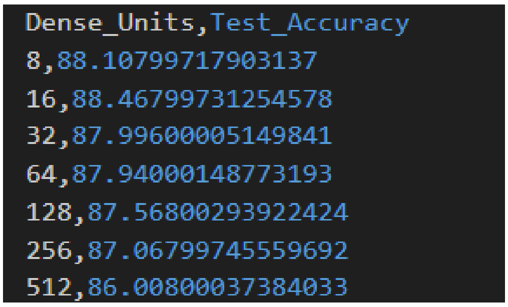
Code:



```python
# ============================================================================
# QUESTION 7: MLP - Increase FC layer neurons
# ============================================================================

print("\n" + "="*80)
print("QUESTION 7: MLP Dense Layer Size Experiments")
print("="*80)

dense_units = [8, 16, 32, 64, 128, 256, 512]
mlp_dense_results = []

for units in dense_units:
    print(f"\nTesting MLP with {units} dense units")
    model = create_mlp_model(dense_units=units)
    accuracy, _ = train_and_evaluate_model(
        model, f"Question 7 - MLP {units} Units", epochs=20, save_plots=False
    )
    mlp_dense_results.append(accuracy)

# Create results dataframe and plot
mlp_dense_df = pd.DataFrame({
    'Dense_Units': dense_units,
    'Test_Accuracy': mlp_dense_results
})
mlp_dense_df.to_csv('question_7_mlp_dense_results.csv', index=False)

# Plot results
plt.figure(figsize=(10, 6))
plt.plot(dense_units, mlp_dense_results, 'go-', linewidth=2, markersize=8)
plt.title('Question 7: MLP Performance vs Dense Layer Size')
plt.xlabel('Number of Dense Units')
plt.ylabel('Test Accuracy (%)')
plt.grid(True, alpha=0.3)
plt.xscale('log')
plt.xticks(dense_units, dense_units)
for i, acc in enumerate(mlp_dense_results):
    plt.annotate(f'{acc:.2f}%', (dense_units[i], acc),
                 textcoords="offset points", xytext=(0,10), ha='center')
plt.savefig('question_7_mlp_dense_comparison.png', dpi=300, bbox_inches='tight')
plt.close()

best_dense_units = dense_units[np.argmax(mlp_dense_results)]
best_dense_accuracy = max(mlp_dense_results)
print(f"\nBest MLP dense units: {best_dense_units} with accuracy: {best_dense_accuracy:.2f}%")
print(f"Results saved to question_7_mlp_dense_results.csv and question_7_mlp_dense_comparison.png")
```

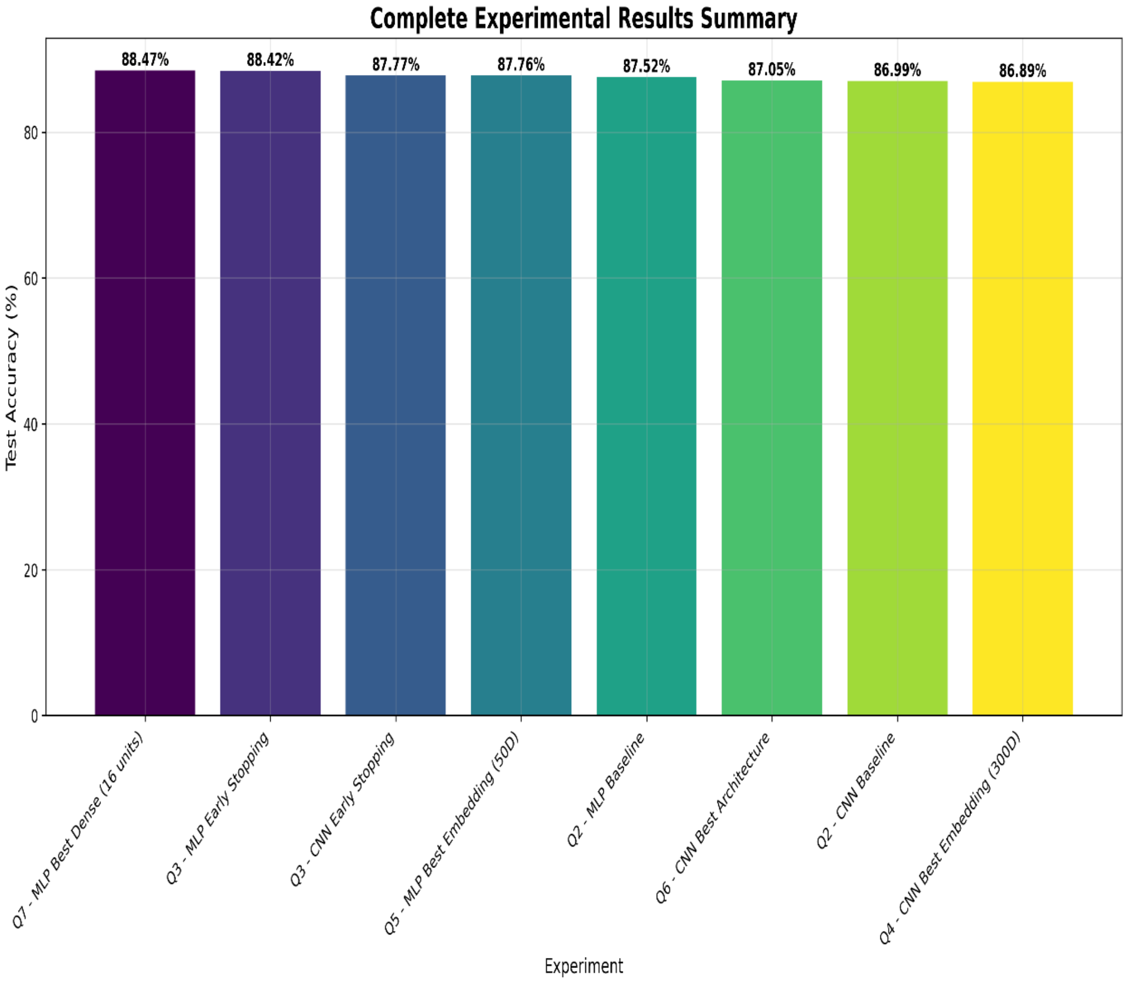
Output for Q-7:

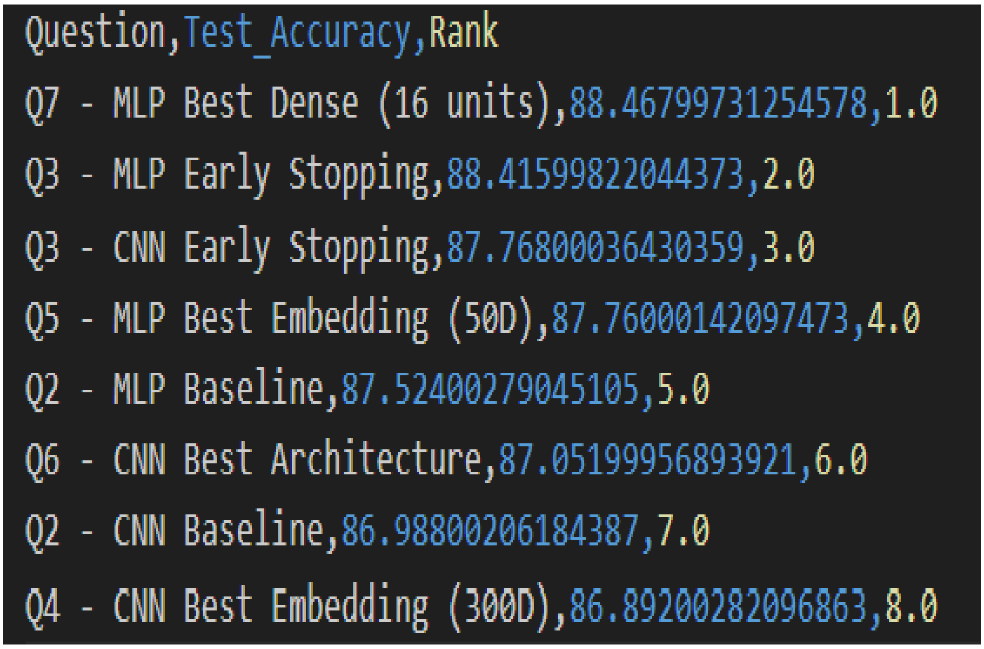




```
Dense_Units,Test_Accuracy
8,88.10799717903137
16,88.46799731254578
32,87.99600005149841
64,87.94000148773193
128,87.56800293922424
256,87.06799745559692
512,86.00800037384033
```

SUMMARY OF ALL EXPERIMENTS





```
Question,Test_Accuracy,Rank
Q7 - MLP Best Dense (16 units),88.46799731254578,1.0
Q3 - MLP Early Stopping,88.41599822044373,2.0
Q3 - CNN Early Stopping,87.76800036430359,3.0
Q5 - MLP Best Embedding (50D),87.76000142097473,4.0
Q2 - MLP Baseline,87.52400279045105,5.0
Q6 - CNN Best Architecture,87.05199956893921,6.0
Q2 - CNN Baseline,86.98800206184387,7.0
Q4 - CNN Best Embedding (300D),86.89200282096863,8.0
```