5.1. (5 pages) Overfitting

What is overfitting.

Overfitting is when a machine learning model learns **too much detail from the training data** — including random noise or patterns that don't generalize — instead of learning the underlying trend.

As a result:

- The model performs **very well on training data**.
- But it performs **poorly on new, unseen data** (validation/test data).

Example

- Suppose you want to predict house prices.
- A well-fit model might learn relationships like: *"larger houses usually cost more."*
- An **overfit** model might memorize quirks, like *"a house with exactly 3.7 bathrooms and blue walls costs $500,000."* That rule works for one house in the training set, but fails for new houses.

Symptoms of Overfitting

- Training accuracy is high, but validation/test accuracy is much lower.
- Model complexity is unnecessarily high (too many parameters/features).

Copy image from textbook_2 and explain

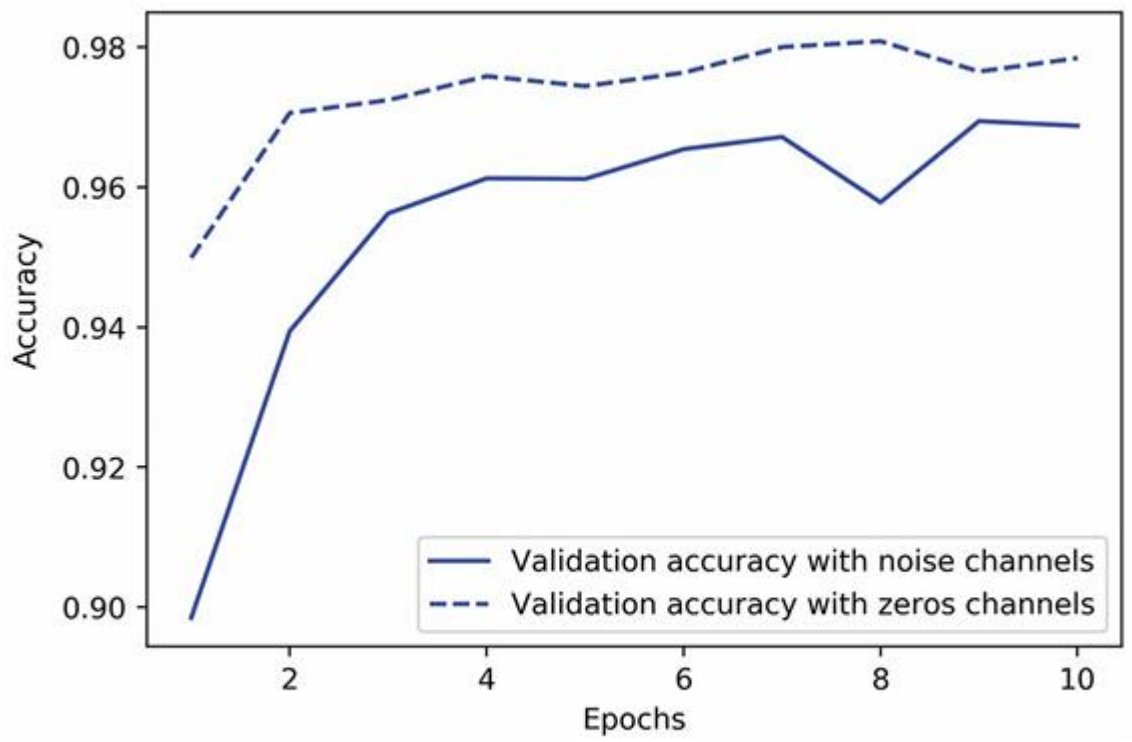


Figure 5.6 Effect of noise channels on validation accuracy

What the figure shows

- **X-axis (horizontal):** Training epochs (how many times the model has seen the data).
- **Y-axis (vertical):** Validation accuracy (how well the model performs on unseen data).
- **Two curves:**
    - **Solid line:** Validation accuracy when extra **noise channels** are added to the input.
    - **Dashed line:** Validation accuracy when extra channels are filled with **zeros** (no useful information, but also not random noise).

Key observations

1. **Early epochs (1–3):**
    - Both models improve rapidly.
    - The model with zero channels (dashed line) achieves slightly higher accuracy early.
    - The noise channels make learning harder initially, since the model wastes effort trying to interpret random signals.
2. **Later epochs (4–10):**
    - Both models plateau around ~0.97–0.98 accuracy.
    - The zero-channel model (dashed) consistently stays a little higher.
    - The noise-channel model (solid) fluctuates more — noise makes generalization harder.

Interpretation

- **Noise channels act like distractions**: The model may overfit to random patterns, hurting validation accuracy.
- **Zero channels are harmless**: They don't add information, but they also don't mislead the model.
- The takeaway: **Feeding irrelevant or noisy features reduces validation accuracy**, since the model might overfit to them.

✅ In short:
This figure demonstrates that adding noisy input features can **hurt validation accuracy** compared to adding empty (zeroed-out) features, even when the model has enough capacity to learn.

Techniques for decreasing/preventing overfitting: present each technique in detail

1. Get More Training Data

- **Why it helps**: Overfitting happens when the model memorizes the training set. More data gives the model more diverse examples, making it harder to memorize and forcing it to learn general patterns.
- **How to do it**:
    - Collect more real-world samples.

    - Use **data augmentation** (for images: rotations, flips, crops; for text: synonym replacement, paraphrasing; for audio: noise injection, pitch shift).
- **Example**: In image classification, flipping or rotating images of cats and dogs makes the dataset larger and reduces memorization.

2. Simplify the Model

- **Why it helps**: Very complex models (deep, wide, many parameters) can memorize data. Simpler models generalize better.
- **How to do it**:
    - Reduce the number of layers or neurons.
    - Prune decision trees (limit depth, remove unnecessary branches).
    - Use fewer features (feature selection).
- **Trade-off**: Too simple → risk of **underfitting**. Need balance.

3. Regularization

- **Why it helps**: Regularization penalizes complexity, discouraging the model from relying on any single weight or feature.
- **Types**:
    - **L1 (Lasso)**: Adds penalty proportional to absolute weights. Encourages sparsity (many weights = 0).
    - **L2 (Ridge/Weight Decay)**: Adds penalty proportional to squared weights. Keeps weights small, avoids dominance of any one feature.
    - **Elastic Net**: Combination of L1 + L2.
- **Equation (L2 example)**:

$$L = L_{\text{original}} + \lambda \sum w^2$$

where λ\lambdaλ is the regularization strength.

4. Dropout (for Neural Networks)

- **Why it helps**: Dropout randomly “turns off” neurons during training. This prevents co-adaptation, forcing the network to spread learning across many neurons.
- **How it works**:
    - At each training step, randomly drop a percentage of neurons (e.g., 20–50%).
    - At test time, all neurons are active, but with scaled-down weights.
- **Effect**: Acts like training many smaller networks and averaging them → reduces overfitting.

5. Early Stopping

- **Why it helps**: Training too long makes the model start memorizing noise.
- **How it works**:
    - Track validation loss/accuracy during training.
    - Stop when validation performance stops improving, even if training accuracy is still going up.
- **Practical tip**: Use a “patience” setting (e.g., wait 5 epochs after last improvement before stopping).

6. Cross-Validation

- **Why it helps**: Ensures the model generalizes across different splits of the data. Prevents accidentally fitting quirks of a single training/validation split.
- **How it works**:
    - **k-fold cross-validation**: Split data into kkk folds, train on k−1k-1k−1, validate on 1, and repeat.
    - Average results to get a stable estimate of generalization.

7. Reduce Input Noise / Irrelevant Features

- **Why it helps**: Noisy or irrelevant features can trick the model into finding fake patterns.

- **How to do it**:
    - Feature selection (remove useless inputs).
    - Dimensionality reduction (PCA, autoencoders).
    - Clean data (remove mislabeled or corrupted samples).
- **Example**: In medical data, patient ID numbers shouldn't be used as features — they carry no predictive meaning but might cause overfitting.

8. Ensemble Methods

- **Why it helps**: Combines multiple models to average out individual overfitting behaviors.
- **Types**:
    - **Bagging (Bootstrap Aggregation)**: Train multiple models on random subsets (e.g., Random Forest).
    - **Boosting**: Sequentially train weak learners that correct previous mistakes (e.g., XGBoost, AdaBoost).
    - **Stacking**: Train different models and combine their outputs with another model.
- **Effect**: Reduces variance, improves robustness.

9. Data Noise Injection (Regularization via Augmentation)

- **Why it helps**: Adding controlled noise during training forces the model to generalize better.
- **Examples**:
    - Add Gaussian noise to inputs.
    - Randomly mask out input values.
    - Jitter labels slightly (in some reinforcement learning settings).

10. Batch Normalization

- **Why it helps**: Normalizes activations in each layer → smoother optimization, less dependency on initial weights, more robust training.
- **Indirect effect**: Also has a regularization-like effect that reduces overfitting.

✅ **Summary**:

- **Data-focused methods**: More data, augmentation, cleaning, feature selection.
- **Model-focused methods**: Simplify, dropout, regularization, early stopping, batch norm.
- **Evaluation-focused methods**: Cross-validation, ensembling.

Examples with codes

1. Regularization (L1 & L2)

```
# 1. Regularization (L1 & L2)
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate toy dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=10, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Logistic Regression with L2 regularization
model = LogisticRegression(penalty="l2", C=0.1, solver="lbfgs", max_iter=1000)
model.fit(X_train, y_train)

print("Train Accuracy:", model.score(X_train, y_train))
print("Validation Accuracy:", model.score(X_val, y_val))
```

```
Train Accuracy: 0.85625
Validation Accuracy: 0.825
```

2. Dropout (Deep Learning)

```python
# 2. Dropout (Deep Learning)
import tensorflow as tf
from tensorflow.keras import layers, models

# Simple Neural Network with Dropout
model = models.Sequential([
    layers.Dense(128, activation="relu", input_shape=(20,)),
    layers.Dropout(0.5),   # 50% of neurons dropped during training
    layers.Dense(64, activation="relu"),
    layers.Dropout(0.3),
    layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
```

Dropout forces the network to not rely on specific neurons → reduces overfitting.

3. Early Stopping

```python
# 3. Early Stopping
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor="val_loss", patience=3, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    epochs=50,
    validation_data=(X_val, y_val),
    callbacks=[early_stop]
)
```

Training will stop if validation loss does not improve for 3 epochs.

4. Data Augmentation (Images)

```python
# 4. Data Augmentation (Images)
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.2
)

# Example: augment images
for X_batch, y_batch in datagen.flow(X_train.reshape(-1,28,28,1), y_train, batch_size=32):
    # Show one augmented batch
    break
```

This generates new “synthetic” training images to avoid memorization.

5. Cross-Validation

```python
# 5. Cross-Validation
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=42)
scores = cross_val_score(rf, X, y, cv=5)

print("Cross-validation scores:", scores)
print("Average:", scores.mean())
```

```
Cross-validation scores: [0.95  0.94  0.93  0.91  0.935]
Average: 0.933
```

Cross-validation ensures the model generalizes across different splits.

6. Ensemble (Bagging)

```python
# 6. Ensemble (Bagging)
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

base_model = DecisionTreeClassifier(max_depth=5)

# Use 'estimator' instead of 'base_estimator'
ensemble = BaggingClassifier(estimator=base_model, n_estimators=20, random_state=42)
ensemble.fit(X_train, y_train)

print("Validation Accuracy:", ensemble.score(X_val, y_val))
```

```
Validation Accuracy: 0.885
```

Bagging reduces variance by averaging multiple models trained on random subsets.

7. Noise Injection (Regularization trick)

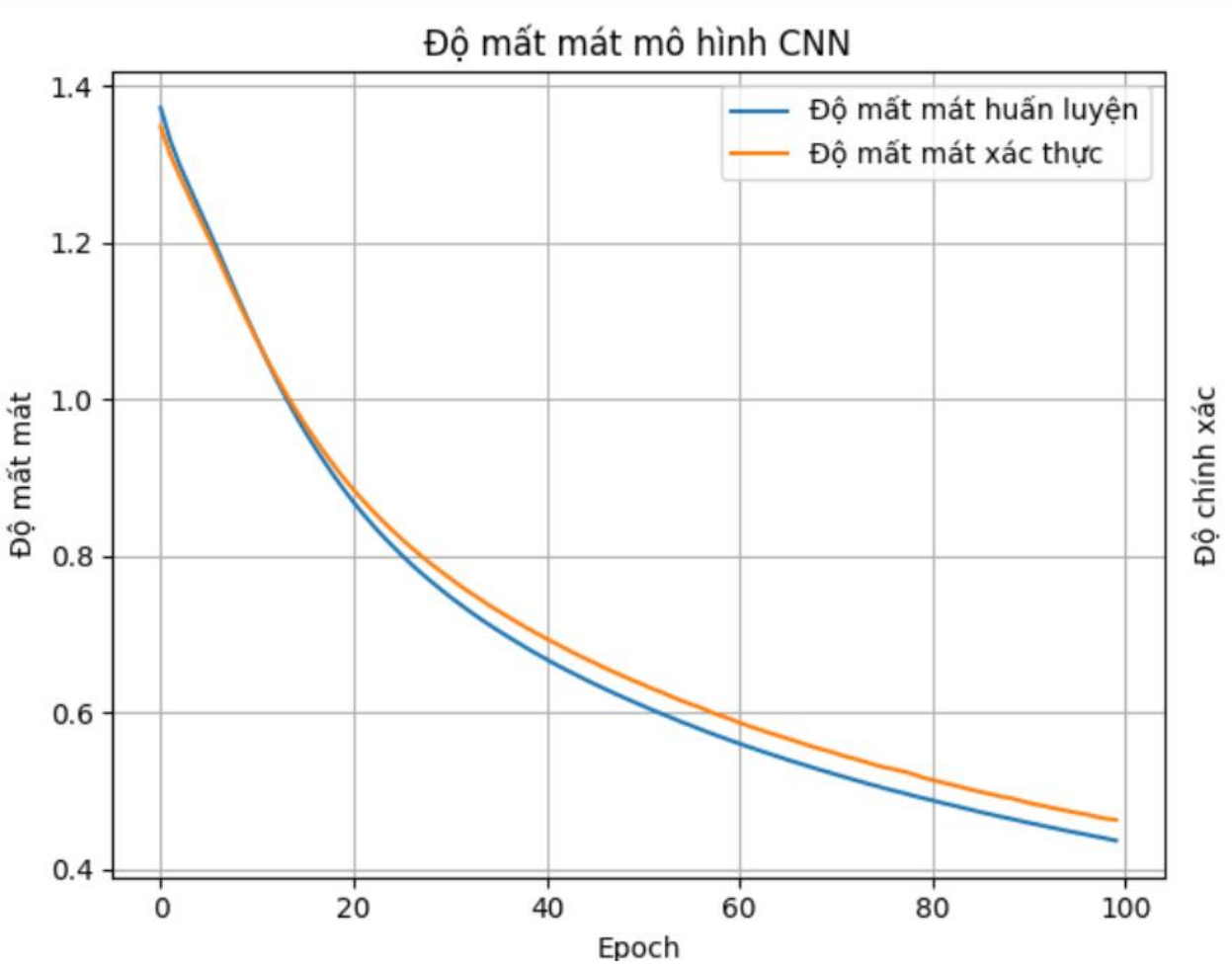
```python
# 7. Noise Injection (Regularization trick)
import numpy as np

# Add Gaussian noise to training data
noise = np.random.normal(0, 0.1, X_train.shape)
X_train_noisy = X_train + noise

# Train with noisy inputs
model.fit(X_train_noisy, y_train, validation_data=(X_val, y_val), epochs=10)
```

Adding small noise to inputs makes the model more robust.

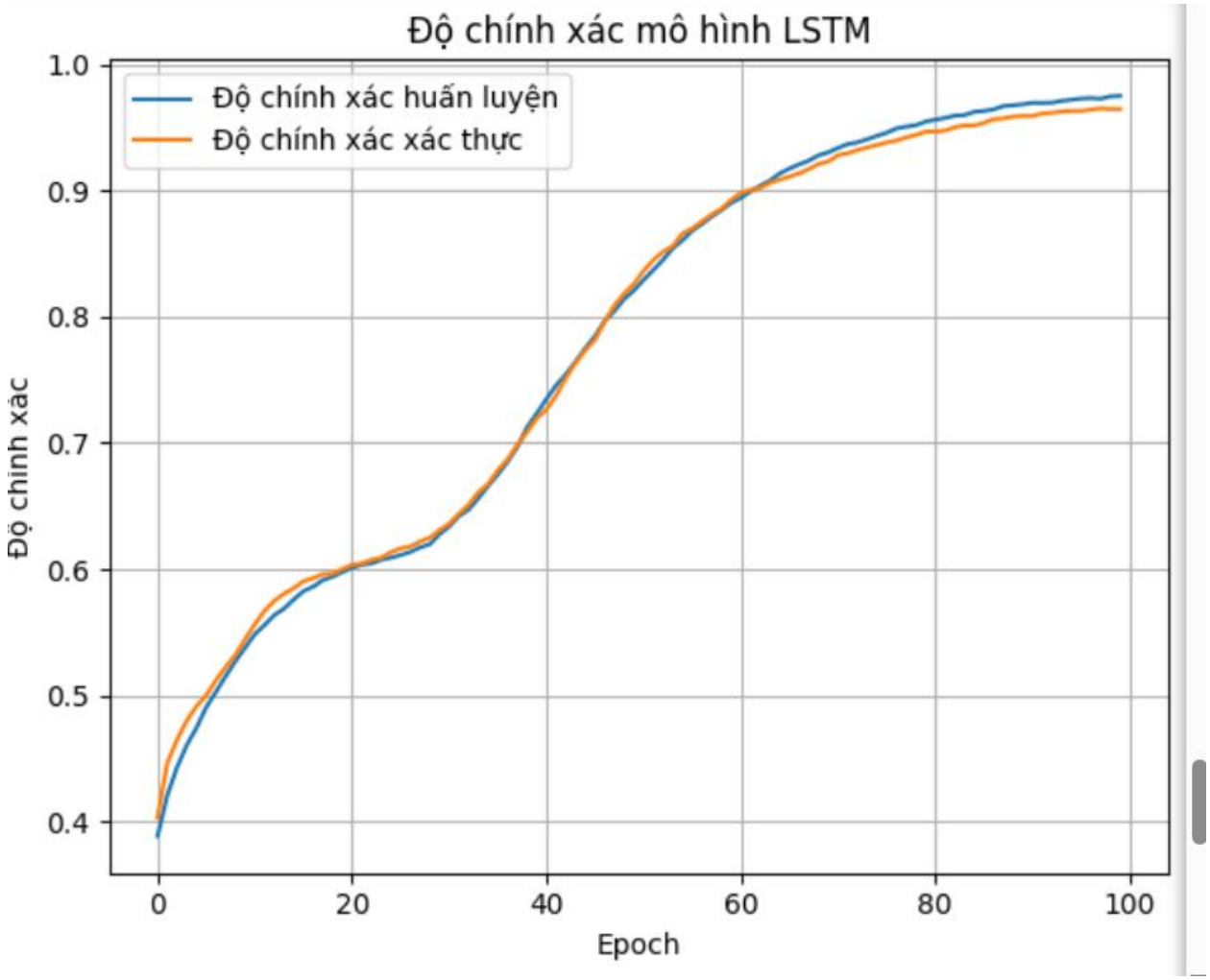
5.2. Using 2 techniques change learning rate, augmentation to decrease overfitting of models

Change learning rate:

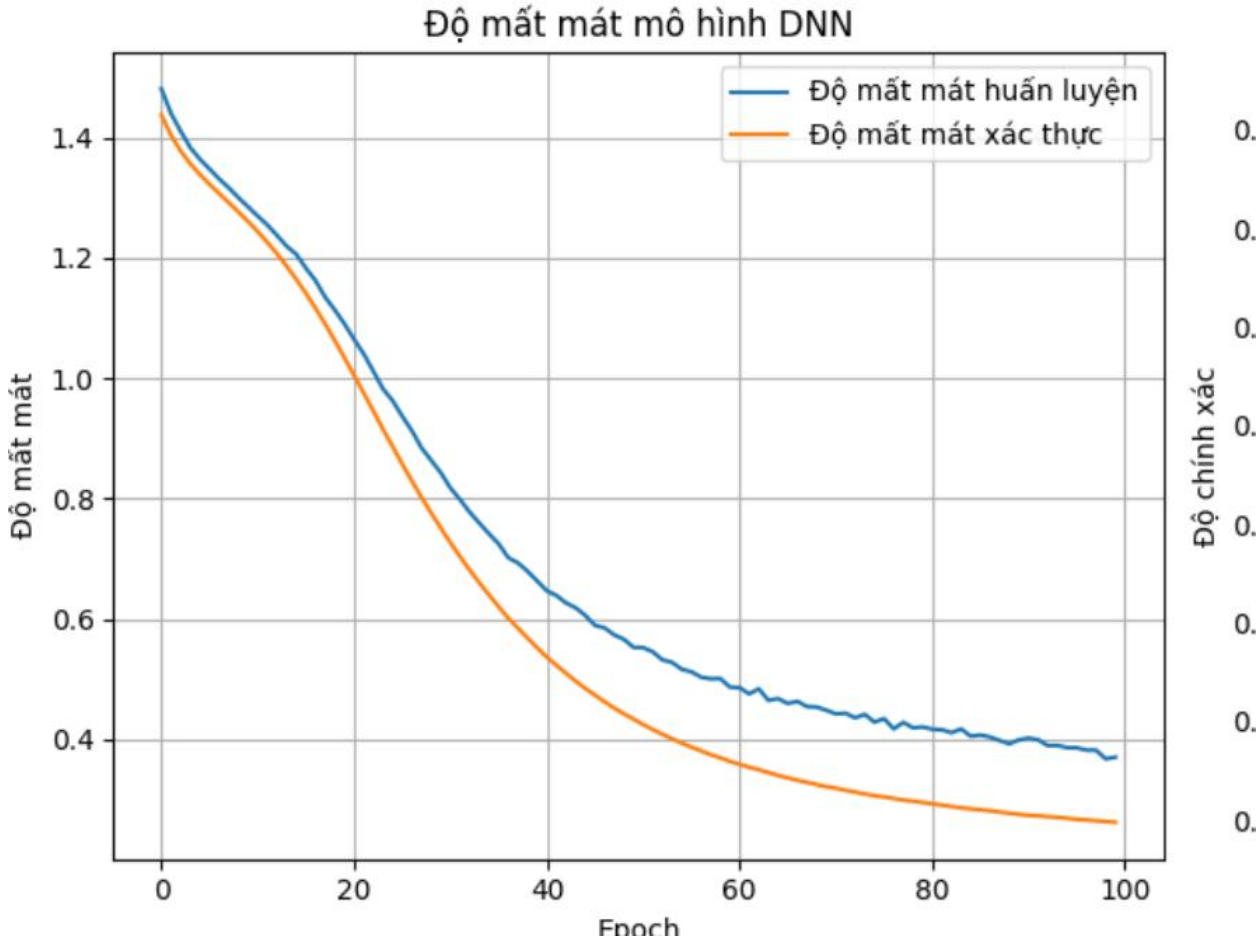
Learning rate = 0.00001



Với mô hình CNN thì từ epoch 20 loss của tập huấn luyện > loss của tập xác thực -> Mô hình bắt đầu bị overfitting

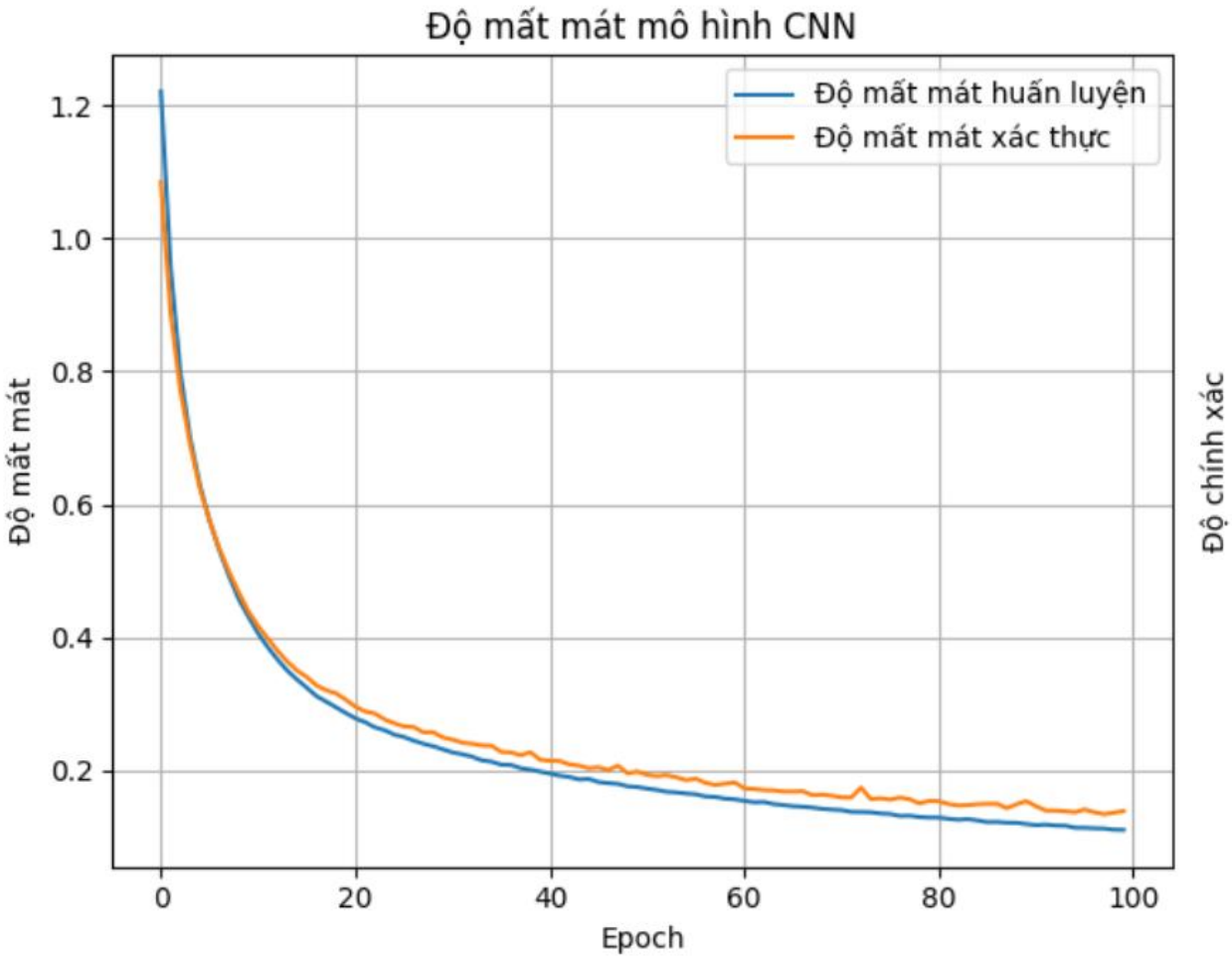


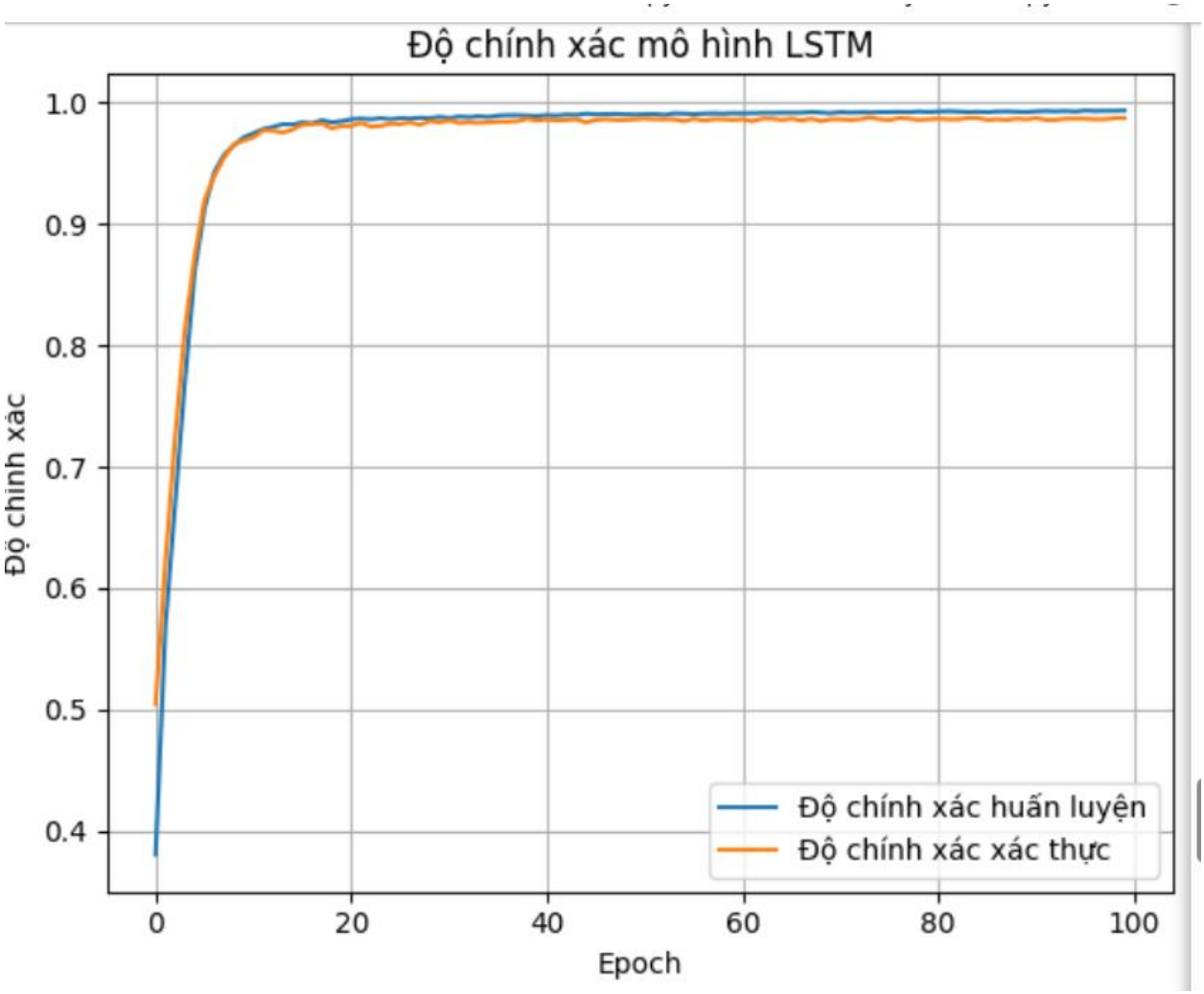
Với mô hình LSTM thì từ epoch khoảng 62 độ chính xác của tập huyến luyện cao hơn độ chính xác tập xác thực -> Mô hình bị overfitting



Loss của tập train > Loss của tập val -> mô hình không bị overfitting

Thay learning rate = 0.0001


Độ mất mát mô hình CNN
Độ mất mát huấn luyện
Độ mất mát xác thực
Độ mất mát
Độ chính xác
Epoch
1.2
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100



Mô hình CNN và LSTM đã không còn bị overfitting nữa

5.3. (5 pages) Presenting 10 techniques for representing text in vector/tensor

```
# Sample corpus
corpus = [
    "The cat sat on the mat",
    "The dog barked at the cat",
    "Dogs and cats are friends"
]
```

1. One-Hot Encoding (Word Counting)

One-hot encoding is the most basic vector representation for text. Each unique word in the vocabulary is assigned an index, and a word is represented by a binary vector where only the index corresponding to that word is 1, and all other entries are 0.

- Vocabulary: {cat, dog, mouse}
- One-hot vector for “dog”: [0, 1, 0]

Advantages: Simple, deterministic.
Disadvantages: Very sparse vectors, no semantic similarity between words.

```
# 1. One-Hot Encoding
from sklearn.preprocessing import OneHotEncoder
import numpy as np

vocab = list(set(" ".join(corpus).lower().split()))
onehot = OneHotEncoder(sparse_output=False)
onehot.fit(np.array(vocab).reshape(-1,1))

print("1. One-hot vector for 'cat':")
print(onehot.transform([["cat"]]))
```

```
1. One-hot vector for 'cat':
[[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

2. Bag-of-Words (BoW)

The Bag-of-Words model represents a document as a vector of word counts, disregarding word order but capturing frequency information.

Example with vocabulary {cat, dog, mouse}:

- Document: “dog dog cat”
- Vector: [1, 2, 0]

Advantages: Easy to compute, works well for simple tasks.
Disadvantages: Ignores word order, results in high-dimensional sparse vectors.

```python
# 2. Bag-of-Words
from sklearn.feature_extraction.text import CountVectorizer
bow = CountVectorizer()
X_bow = bow.fit_transform(corpus)
print("\n2. Bag-of-Words representation (doc x vocab):")
print(X_bow.toarray())
print("Vocabulary:", bow.get_feature_names_out())
```

```
2. Bag-of-Words representation (doc x vocab):
[[0 0 0 0 1 0 0 0 0 1 1 1 2]
 [0 0 1 1 1 0 1 0 0 0 0 0 2]
 [1 1 0 0 0 1 0 1 1 0 0 0 0]]
Vocabulary: ['and' 'are' 'at' 'barked' 'cat' 'cats' 'dog' 'dogs' 'friends' 'mat' 'on'
 'sat' 'the']
```

3. Term Frequency – Inverse Document Frequency (TF-IDF)

TF-IDF improves upon BoW by weighting words according to their importance in a document relative to the entire corpus.

$$TFIDF(t,d) = TF(t,d) \times \log \frac{N}{DF(t)}$$

where:

- $TF(t,d)$: frequency of term t in document d
- N: total number of documents
- $DF(t)$: number of documents containing term t

Advantages: Reduces the weight of common words (e.g., “the”, “and”).
Disadvantages: Still ignores semantics and word order.

```
# 3. TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
X_tfidf = tfidf.fit_transform(corpus)
print("\n3. TF-IDF representation:")
print(X_tfidf.toarray())
print("Vocabulary:", tfidf.get_feature_names_out())
```

```
3. TF-IDF representation:
[[0.         0.         0.         0.         0.31331607 0.
  0.         0.         0.         0.41197298 0.41197298 0.41197298
  0.62663214]
 [0.         0.         0.41197298 0.41197298 0.31331607 0.
  0.41197298 0.         0.         0.         0.         0.
  0.62663214]
 [0.4472136  0.4472136  0.         0.         0.         0.4472136
  0.         0.4472136  0.4472136  0.         0.         0.
  0.        ]]
Vocabulary: ['and' 'are' 'at' 'barked' 'cat' 'cats' 'dog' 'dogs' 'friends' 'mat' 'on'
 'sat' 'the']
```

4. Word2Vec (Mikolov et al., 2013)

Word2Vec uses a shallow neural network to learn word embeddings such that words with similar meanings are close in vector space. Two main training architectures exist: **Skip-gram** and **Continuous Bag-of-Words (CBOW)**.

- Skip-gram predicts context words from a target word.

- CBOW predicts a target word from surrounding context words.

Example: “king – man + woman ≈ queen”

Advantages: Captures semantic and syntactic relationships.
Disadvantages: Static embeddings; same vector for a word in all contexts.

```python
# 4. Word2Vec (using gensim)
from gensim.models import Word2Vec
sentences = [doc.lower().split() for doc in corpus]
w2v = Word2Vec(sentences, vector_size=50, window=3, min_count=1, sg=1)
print("\n4. Word2Vec embedding for 'cat':")
print(w2v.wv['cat'][:10])  # show first 10 dims
```

```
4. Word2Vec embedding for 'cat':
[-0.01631583  0.0089916  -0.00827415  0.00164907  0.01699724 -0.00892435
  0.009035   -0.01357392 -0.00709698  0.01879702]
```

5. GloVe (Global Vectors for Word Representation)

GloVe combines the advantages of matrix factorization and predictive models. It uses word co-occurrence statistics from a large corpus to learn dense embeddings.

Objective function (simplified):

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$

where X_{ij} is co-occurrence count of words i and j.

Advantages: Incorporates global corpus statistics.
Disadvantages: Still produces static embeddings.

```python
# 5. GloVe (via gensim downloader)
import gensim.downloader as api
glove = api.load("glove-wiki-gigaword-50")
print("\n5. GloVe embedding for 'cat':")
print(glove['cat'][:10])
```

```
5. GloVe embedding for 'cat':
[ 0.45281  -0.50108  -0.53714  -0.015697  0.22191   0.54602  -0.67301
 -0.6891    0.63493  -0.19726 ]
```

6. FastText

FastText (Facebook AI) improves Word2Vec by considering subword information (character n-grams). This allows it to handle rare and out-of-vocabulary words.

- Example: “playing” → subwords: “play”, “lay”, “ing”
- The final embedding is the sum of embeddings of its subwords.

Advantages: Handles morphology, useful for morphologically rich languages.
Disadvantages: Still context-independent.

```python
# 6. FastText
from gensim.models.fasttext import FastText
ft = FastText(sentences, vector_size=50, window=3, min_count=1)
print("\n6. FastText embedding for 'cats':")
print(ft.wv['cats'][:10])  # works for plural form
```

```
6. FastText embedding for 'cats':
[-2.3170598e-03 -2.0539537e-03  1.0790244e-03 -6.1491965e-03
  5.3702937e-03 -2.6794572e-04 -7.4717490e-04 -2.7384354e-05
  2.9085550e-04  2.4583337e-03]
```

7. ELMo (Embeddings from Language Models)

ELMo (Peters et al., 2018) generates **contextual word embeddings** using deep bidirectional LSTMs trained on a language modeling objective. Each word's representation depends on the entire sentence.

- Example: "bank" in "river bank" vs. "bank loan" → different vectors.

Advantages: Context-aware, captures polysemy.
Disadvantages: Computationally heavy, replaced by transformers in practice.

```
# 7. ELMo (via allennlp)
# from allennlp.commands.elmo import ElmoEmbedder
# elmo = ElmoEmbedder()
# tokens = ["the", "cat", "sat"]
# embeddings = elmo.embed_sentence(tokens)
# print("\n7. ELMo embedding for 'cat' (dim):", embeddings.shape)
```

8. BERT (Bidirectional Encoder Representations from Transformers)

BERT (Devlin et al., 2019) uses a transformer encoder to generate deep contextual embeddings. It is pretrained using **Masked Language Modeling (MLM)** and **Next Sentence Prediction (NSP)**.

- Input: "The cat sat on the [MASK]."
- Output: predicts "mat".

Advantages: State-of-the-art performance across NLP tasks.
Disadvantages: Requires large-scale pretraining, resource intensive.

```
# 8. BERT
from transformers import BertTokenizer, BertModel
import torch

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertModel.from_pretrained("bert-base-uncased")
inputs = tokenizer("The cat sat on the mat", return_tensors="pt")
outputs = model(**inputs)
print("\n8. BERT embedding shape:", outputs.last_hidden_state.shape)
```

```
8. BERT embedding shape: torch.Size([1, 8, 768])
WARNING:tensorflow:From C:\Users\nvqua\AppData\Local\Programs\Python\Python311\Lib\site-packages\tf_keras\src\losses.py:2976: The name tf.losses.sparse_
softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

9. Sentence Embeddings (Doc2Vec, Sentence-BERT)

While word embeddings capture word-level meaning, document and sentence embeddings map entire sequences into a single vector.

- **Doc2Vec** (Le & Mikolov, 2014): Adds a paragraph vector to Word2Vec framework.
- **Sentence-BERT**: Fine-tunes BERT using a Siamese architecture to produce semantically meaningful sentence embeddings.

Advantages: Useful for semantic similarity and retrieval tasks.
Disadvantages: Training and fine-tuning can be resource-heavy.

```python
# 9. Sentence-BERT (sentence embeddings)
from sentence_transformers import SentenceTransformer
sbert = SentenceTransformer("all-MiniLM-L6-v2")
sent_emb = sbert.encode("The cat sat on the mat")
print("\n9. Sentence-BERT embedding (dim):", sent_emb.shape)
```

```
9. Sentence-BERT embedding (dim): (384,)
```

10. Transformer-based Large Language Models (GPT, T5, etc.)

Modern LLMs such as GPT and T5 provide contextualized embeddings at the token, sentence, and document levels. The hidden states of transformer layers serve as rich representations for downstream tasks.

Advantages: Extremely powerful, transfer learning across tasks.
Disadvantages: Very large models, require significant compute and storage.

```python
# 10. GPT-like embeddings (OpenAI or HuggingFace models)
from transformers import AutoTokenizer, AutoModel
tok = AutoTokenizer.from_pretrained("gpt2")
mdl = AutoModel.from_pretrained("gpt2")
inp = tok("The cat sat on the mat", return_tensors="pt")
out = mdl(**inp)
print("\n10. GPT2 embedding shape:", out.last_hidden_state.shape)
```

```
10. GPT2 embedding shape: torch.Size([1, 6, 768])
```

Conclusion

Text representation has evolved from simple sparse encodings (One-hot, BoW, TF-IDF) to dense neural embeddings (Word2Vec, GloVe, FastText) and finally to **contextual embeddings** from deep language models (ELMo, BERT, GPT). Each method offers a trade-off between simplicity, interpretability, and semantic richness. The choice of representation depends on the task, resources, and desired accuracy.

5.4

```
# Tệp này minh họa cách phân tích cảm xúc
# trong đánh giá phim bằng cách sử dụng Word2Vec, CNN và LSTM.
# Nguyen Viet Quang
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Conv1D, LSTM, Dense, Dropout, Input, SimpleRNN
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from gensim.models import Word2Vec
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import matplotlib.pyplot as plt

# Tải các tài nguyên cần thiết cho nltk
# Nếu bạn chưa có, hãy bỏ chú thích và chạy các lệnh này lần đầu
# nltk.download('punkt')
# nltk.download('stopwords')

# --- 1. Tải tập dữ liệu ---
print("1. Đang tải bộ dữ liệu đánh giá phim...")
```

```
# --- Tùy chọn: Tải dữ liệu từ file CSV của bạn ---
# Bỏ chú thích dòng dưới đây và thay đổi đường dẫn file nếu bạn có tệp CSV.
file_path = 'filmReview_1.csv'
try:
    df = pd.read_csv(file_path)
    print(f"Đã tải thành công bộ dữ liệu từ '{file_path}'.")
except FileNotFoundError:
    # --- Tùy chọn: Sử dụng dữ liệu giả lập (mặc định) ---
    # Dữ liệu này được sử dụng để minh họa vì không thể truy cập file cục bộ.
    print(f"Không tìm thấy file '{file_path}'. Sử dụng bộ dữ liệu giả lập để minh họa...")
    data = {
        'review_id': range(1, 11),
        'user_id': ['user_001', 'user_002', 'user_003', 'user_004', 'user_005', 'user_006', 'user_007', 'user_008', 'user_009', 'user_010'],
        'user_name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank', 'Grace', 'Heidi', 'Ivan', 'Judy'],
        'film_title': ['Spirited Away', 'The Shawshank Redemption', 'Pulp Fiction', 'The Dark Knight', 'Inception', 'Forrest Gump', 'Parasite', 'Fight C
        'rating': [10, 2, 5, 9, 8, 3, 10, 7, 4, 9],
        'review_text': [
            "This is a fantastic movie! The story is very engaging and the acting is top-notch.",
            "A terrible film, I don't understand why so many people praised it. Very disappointed.",
            "Beautiful visuals, great sound, but the plot is too disjointed.",
            "I really like this movie, I've watched it 3 times.",
            "Nothing special. It was okay to watch but not impressive.",
            "A waste of time. The plot is boring and confusing.",
            "A very worthwhile film, a true work of art!",
            "It's a humorous film but with depth. Very impressive.",
            "Excellent special effects, but I don't like the character development.",
            "A classic cinematic film. Full of emotion.",
        ]
```

```
            "A classic cinematic film. Full of emotion.",
        ],
        'review_date': pd.to_datetime(['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04', '2023-01-05', '2023-01-06', '2023-01-07', '2023-01-08', '
    }
    df = pd.DataFrame(data)

print("Bộ dữ liệu đã được tải.")
print(df.head())
print("-" * 50)

# --- 2. Tiền xử lý dữ liệu và tạo nhãn cảm xúc ---
# Chuyển đổi điểm rating thành nhãn cảm xúc nhị phân: 1 (tích cực) hoặc 0 (tiêu cực)
# Giả định: rating > 5 là tích cực, <= 5 là tiêu cực
df['sentiment'] = df['rating'].apply(lambda x: 1 if x > 5 else 0)

# Tiền xử lý văn bản cho Word2Vec
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)  # Xóa dấu câu
    words = word_tokenize(text)
    return words

df['preprocessed_text'] = df['review_text'].apply(preprocess_text)
review_sentences = df['preprocessed_text'].tolist()
labels = tf.convert_to_tensor(df['sentiment'].values, dtype=tf.int32)
```

```
print("Hoàn tất tiền xử lý văn bản.")
print("-" * 50)

# --- 3. Huấn luyện mô hình Word2Vec ---
print("3. Đang huấn luyện mô hình Word2Vec...")

embedding_dim = 100 # Kích thước vector biểu diễn
word2vec_model = Word2Vec(
    sentences=review_sentences,
    vector_size=embedding_dim,
    window=5,
    min_count=1,
    workers=4
)

# Tạo vector biểu diễn cho mỗi đánh giá bằng cách lấy trung bình các vector từ
def get_sentence_vector(sentence, model):
    word_vectors = [model.wv[word] for word in sentence if word in model.wv]
    if len(word_vectors) == 0:
        return np.zeros(model.vector_size)
    return np.mean(word_vectors, axis=0)

X_vectors = np.array([get_sentence_vector(s, word2vec_model) for s in review_sentences])
# Reshape để phù hợp với các lớp mô hình
X_vectors = X_vectors.reshape(X_vectors.shape[0], 1, X_vectors.shape[1])
```

```python
print("Hoàn tất tạo vector biểu diễn Word2Vec.")
print("-" * 50)

# --- 4. Xây dựng và Huấn luyện các mô hình ---

def build_and_train_model(model_name, model_fn):
    print(f"4. Đang xây dựng và huấn luyện mô hình: {model_name}...")
    model = model_fn()
    model.summary()

    model.compile(
        optimizer=Adam(learning_rate=1e-5),
        loss='binary_crossentropy',
        metrics=['accuracy']
    )

    history = model.fit(
        X_vectors,
        labels,
        epochs=10,
        batch_size=2,
        validation_split=0.2,
        verbose=1
    )
    return model, history
```

```python
# a) Mô hình Word2Vec + LSTM
def create_lstm_model():
    input_layer = Input(shape=(1, embedding_dim), dtype='float32')
    lstm_layer = LSTM(32)(input_layer)
    lstm_layer = Dropout(0.3)(lstm_layer)
    output_layer = Dense(1, activation='sigmoid')(lstm_layer)
    return Model(inputs=input_layer, outputs=output_layer)

lstm_model, lstm_history = build_and_train_model("Word2Vec + LSTM", create_lstm_model)
print("Hoàn tất huấn luyện mô hình LSTM.")
print("-" * 50)

# b) Mô hình Word2Vec + CNN
def create_cnn_model():
    input_layer = Input(shape=(1, embedding_dim), dtype='float32')
    cnn_layer = Conv1D(filters=64, kernel_size=1, activation='relu')(input_layer)
    cnn_layer = Dropout(0.3)(cnn_layer)
    cnn_layer = tf.keras.layers.Flatten()(cnn_layer)
    output_layer = Dense(1, activation='sigmoid')(cnn_layer)
    return Model(inputs=input_layer, outputs=output_layer)

cnn_model, cnn_history = build_and_train_model("Word2Vec + CNN", create_cnn_model)
print("Hoàn tất huấn luyện mô hình CNN.")
print("-" * 50)
```

```python
# c) Mô hình Word2Vec + RNN
def create_rnn_model():
    input_layer = Input(shape=(1, embedding_dim), dtype='float32')
    rnn_layer = SimpleRNN(64)(input_layer)
    rnn_layer = Dropout(0.3)(rnn_layer)
    output_layer = Dense(1, activation='sigmoid')(rnn_layer)
    return Model(inputs=input_layer, outputs=output_layer)

rnn_model, rnn_history = build_and_train_model("Word2Vec + RNN", create_rnn_model)
print("Hoàn tất huấn luyện mô hình RNN.")
print("-" * 50)

# --- 5. Trực quan hóa kết quả huấn luyện ---
def plot_metrics(history, model_name):
    """Vẽ biểu đồ độ chính xác và hàm mất mát của mô hình."""
    plt.style.use('seaborn-v0_8-whitegrid')
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))
    fig.suptitle(f'Kết quả Huấn luyện Mô hình {model_name}', fontsize=16)

    # Biểu đồ Accuracy
    ax1.plot(history.history['accuracy'], label='Độ chính xác huấn luyện')
    ax1.plot(history.history['val_accuracy'], label='Độ chính xác kiểm tra')
    ax1.set_title('Độ chính xác')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Accuracy')
    ax1.legend()
```

```python
    # Biểu đồ Loss
    ax2.plot(history.history['loss'], label='Hàm mất mát huấn luyện')
    ax2.plot(history.history['val_loss'], label='Hàm mất mát kiểm tra')
    ax2.set_title('Hàm mất mát')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Loss')
    ax2.legend()

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()

# Hiển thị biểu đồ cho từng mô hình
plot_metrics(lstm_history, "Word2Vec + LSTM")
plot_metrics(cnn_history, "Word2Vec + CNN")
plot_metrics(rnn_history, "Word2Vec + RNN")
print("Hoàn tất trực quan hóa kết quả huấn luyện.")
print("-" * 50)

# --- 6. Sử dụng các mô hình để dự đoán trên dữ liệu mới ---
print("6. Đang dự đoán trên các đánh giá mới...")
```

```python
new_reviews = [
    "The movie was not good, the plot was illogical.",
    "Excellent film, worth watching many times!",
    "Beautiful visuals, but the content wasn't very special.",
]

# Chuẩn bị dữ liệu mới cho các mô hình
preprocessed_new_reviews = [preprocess_text(r) for r in new_reviews]
new_vectors = np.array([get_sentence_vector(s, word2vec_model) for s in preprocessed_new_reviews])
new_vectors = new_vectors.reshape(new_vectors.shape[0], 1, new_vectors.shape[1])

# Dự đoán và in kết quả cho từng mô hình
def predict_and_print_results(model, model_name):
    predictions = model.predict(new_vectors)
    print(f"\nKết quả dự đoán của mô hình {model_name}:")
    for i, review in enumerate(new_reviews):
        sentiment = "Positive" if predictions[i][0] > 0.5 else "Negative"
        print(f"Đánh giá: '{review}' -> Dự đoán: {sentiment} (Điểm: {predictions[i][0]:.4f})")

predict_and_print_results(lstm_model, "Word2Vec + LSTM")
predict_and_print_results(cnn_model, "Word2Vec + CNN")
predict_and_print_results(rnn_model, "Word2Vec + RNN")

print("-" * 50)
print("Chương trình đã hoàn tất. Bạn có thể thay đổi dữ liệu và mô hình để thử nghiệm thêm.")
```

```
1. Đang tải bộ dữ liệu đánh giá phim...
Đã tải thành công bộ dữ liệu từ 'filmReview_1.csv'.
Bộ dữ liệu đã được tải.
   review_id   user_id          user_name      film_title  rating  \
0          1  user_181       Cô Kim Dương   Phim_title_55       9
1          2  user_271   Phương Thế Hoàng  Phim_title_173       9
2          3  user_124          Anh Dương  Phim_title_162       1
3          4  user_406   Quý ông Trung Vũ   Phim_title_24       6
4          5  user_142  Thành Quang Dương  Phim_title_187       6

                                        review_text review_date
0  This film is a masterpiece! The acting is phen...  2023-12-23
1  This film is a masterpiece! The acting is phen...  2023-10-05
2  Completely disappointed. The plot is weak, and...  2021-03-09
3  An average movie with some good moments, but i...  2023-10-23
4  An average movie with some good moments, but i...  2021-06-27
--------------------------------------------------
Hoàn tất tiền xử lý văn bản.
--------------------------------------------------
3. Đang huấn luyện mô hình Word2Vec...
Hoàn tất tạo vector biểu diễn Word2Vec.
--------------------------------------------------
4. Đang xây dựng và huấn luyện mô hình: Word2Vec + LSTM...


Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 1, 100)	0
lstm (LSTM)	(None, 32)	17,024
dropout (Dropout)	(None, 32)	0
dense (Dense)	(None, 1)	33

Total params: 17,057 (66.63 KB)

Trainable params: 17,057 (66.63 KB)

Non-trainable params: 0 (0.00 B)

---

4. Đang xây dựng và huấn luyện mô hình: Word2Vec + CNN...

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 1, 100)	0
conv1d (Conv1D)	(None, 1, 64)	6,464
dropout_1 (Dropout)	(None, 1, 64)	0
flatten (Flatten)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

Total params: 6,529 (25.50 KB)

Trainable params: 6,529 (25.50 KB)

Non-trainable params: 0 (0.00 B)

4. Đang xây dựng và huấn luyện mô hình: Word2Vec + RNN...

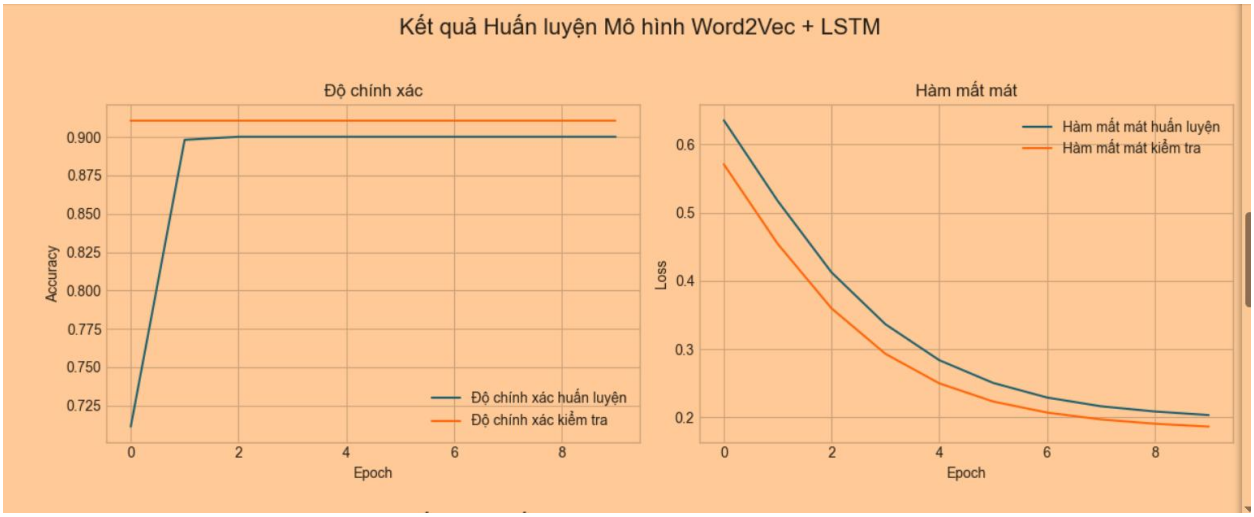
Model: "functional_2"

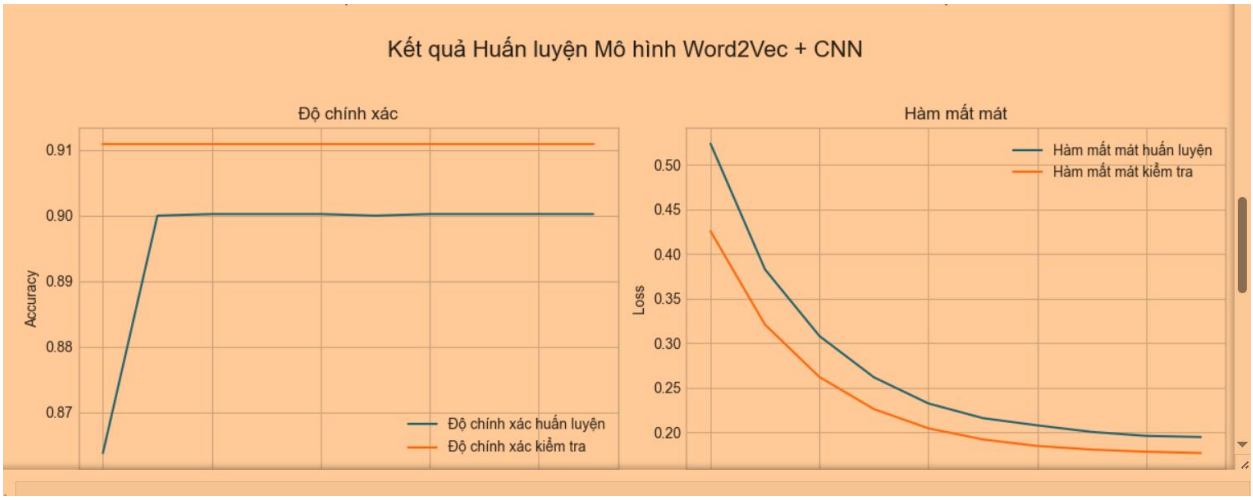
Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 1, 100)	0
simple_rnn (SimpleRNN)	(None, 64)	10,560
dropout_2 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 1)	65

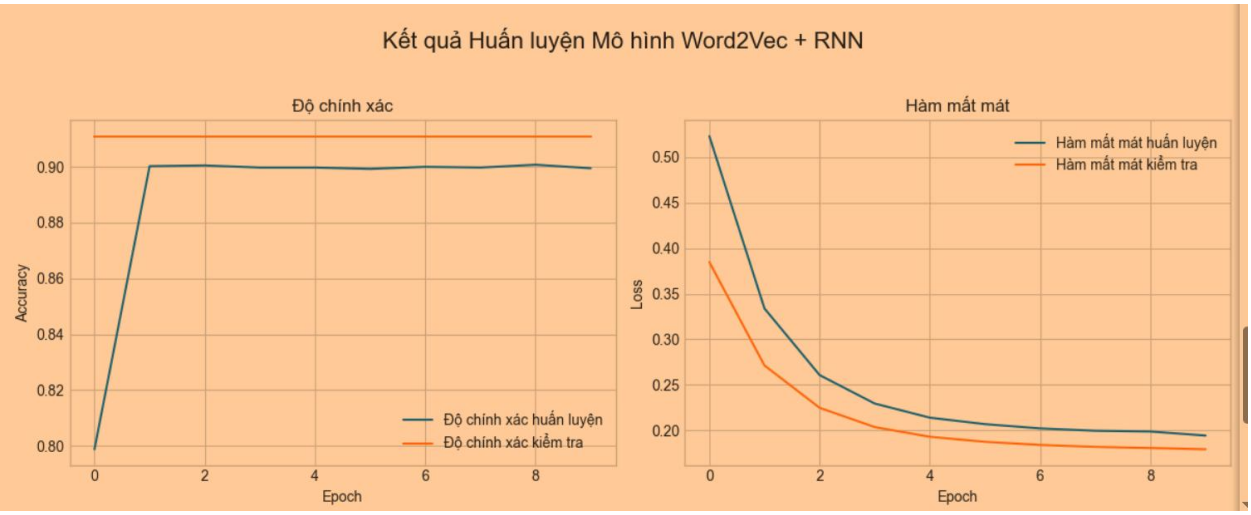
Total params: 10,625 (41.50 KB)

Trainable params: 10,625 (41.50 KB)

Non-trainable params: 0 (0.00 B)








```
Hoàn tất trực quan hóa kết quả huấn luyện.
--------------------------------------------------
6. Đang dự đoán trên các đánh giá mới...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 351ms/step

Kết quả dự đoán của mô hình Word2Vec + LSTM:
Đánh giá: 'The movie was not good, the plot was illogical.' -> Dự đoán: Negative (Điểm: 0.4724)
Đánh giá: 'Excellent film, worth watching many times!' -> Dự đoán: Positive (Điểm: 0.9930)
Đánh giá: 'Beautiful visuals, but the content wasn't very special.' -> Dự đoán: Positive (Điểm: 0.6280)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step

Kết quả dự đoán của mô hình Word2Vec + CNN:
Đánh giá: 'The movie was not good, the plot was illogical.' -> Dự đoán: Negative (Điểm: 0.4193)
Đánh giá: 'Excellent film, worth watching many times!' -> Dự đoán: Positive (Điểm: 0.9998)
Đánh giá: 'Beautiful visuals, but the content wasn't very special.' -> Dự đoán: Positive (Điểm: 0.6038)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 300ms/step

Kết quả dự đoán của mô hình Word2Vec + RNN:
Đánh giá: 'The movie was not good, the plot was illogical.' -> Dự đoán: Negative (Điểm: 0.3312)
Đánh giá: 'Excellent film, worth watching many times!' -> Dự đoán: Positive (Điểm: 0.9990)
Đánh giá: 'Beautiful visuals, but the content wasn't very special.' -> Dự đoán: Negative (Điểm: 0.4835)
--------------------------------------------------
Chương trình đã hoàn tất. Bạn có thể thay đổi dữ liệu và mô hình để thử nghiệm thêm.
```

Các mô hình đều không bị overfitting