# EMAIL SPAM DETECTION

ANGGOTA KELOMPOK:

ANDREW - 2702286715

NICHOLAS WIRA ANGKASA - 2702294521

# PROBLEM

# OVERVIEW

Email spam is one of the most widespread digital threats.
Spam messages often contain phishing links, scam offers, malware, and misleading promotions.
Objectives of this study:

- Build an email spam classifier for the Indonesian language
- Compare two different modeling approaches:
    a. TF-IDF + Convolutional Neural Network (CNN)
    b. IndoBERT + Convolutional Neural Network (CNN)

Key Questions:

1. Can TF-IDF + CNN achieve competitive accuracy for spam detection?
2. Does IndoBERT + CNN significantly improve performance due to contextual embeddings?

# INTRODUCTION

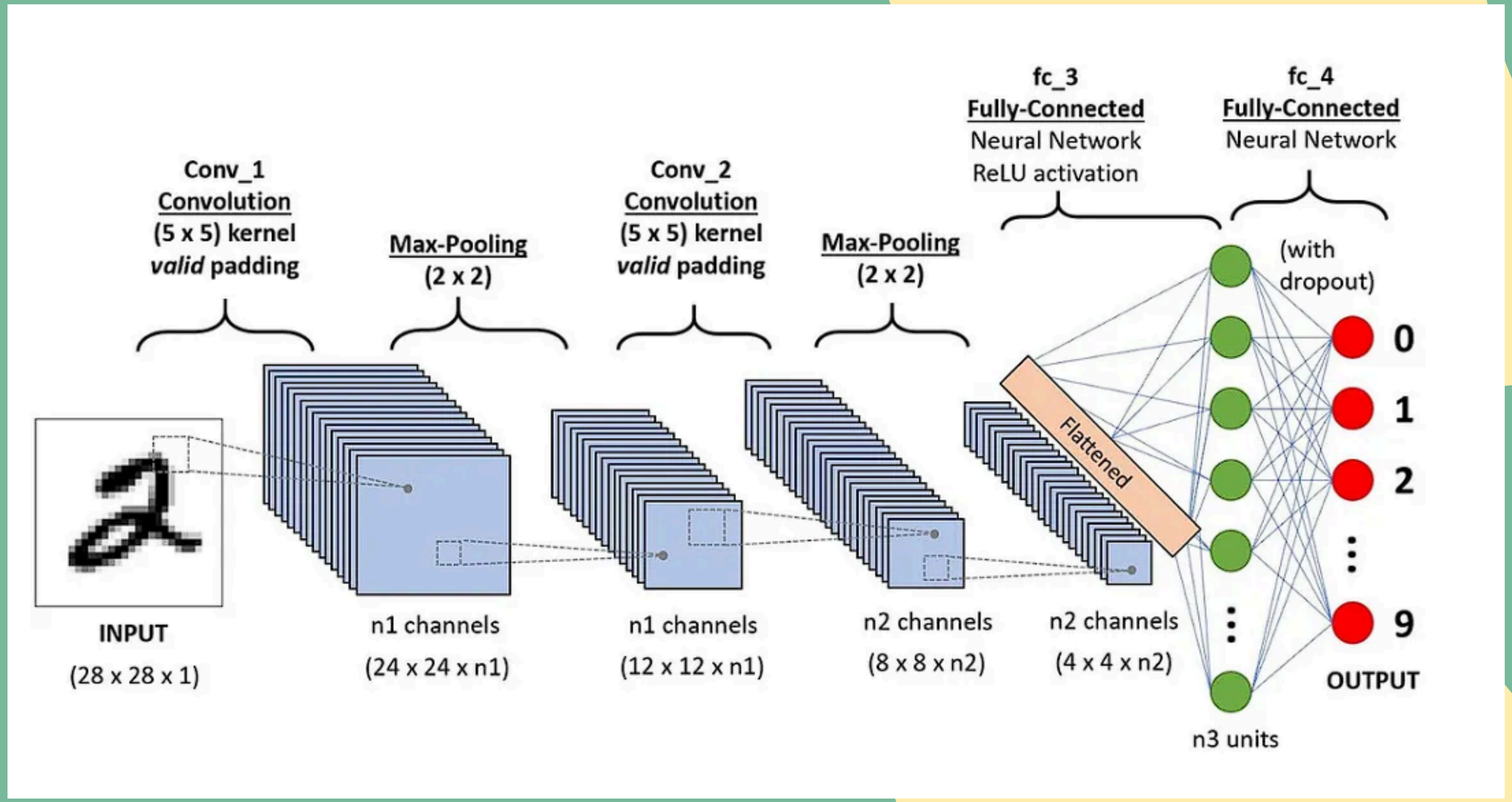
# WHAT IS CNN?

A Convolutional Neural Network (CNN) is a specialized deep learning architecture designed to automatically learn meaningful features directly from raw data. CNNs are widely used in image recognition, object detection, and computer vision, but they are also effective for non-image data such as text, audio, and time-series signals.

The key advantage of CNNs lies in their ability to:

- Capture local patterns
- Preserve spatial or sequential structure
- Reduce the need for manual feature engineering

# CNN ARCHITECTURE

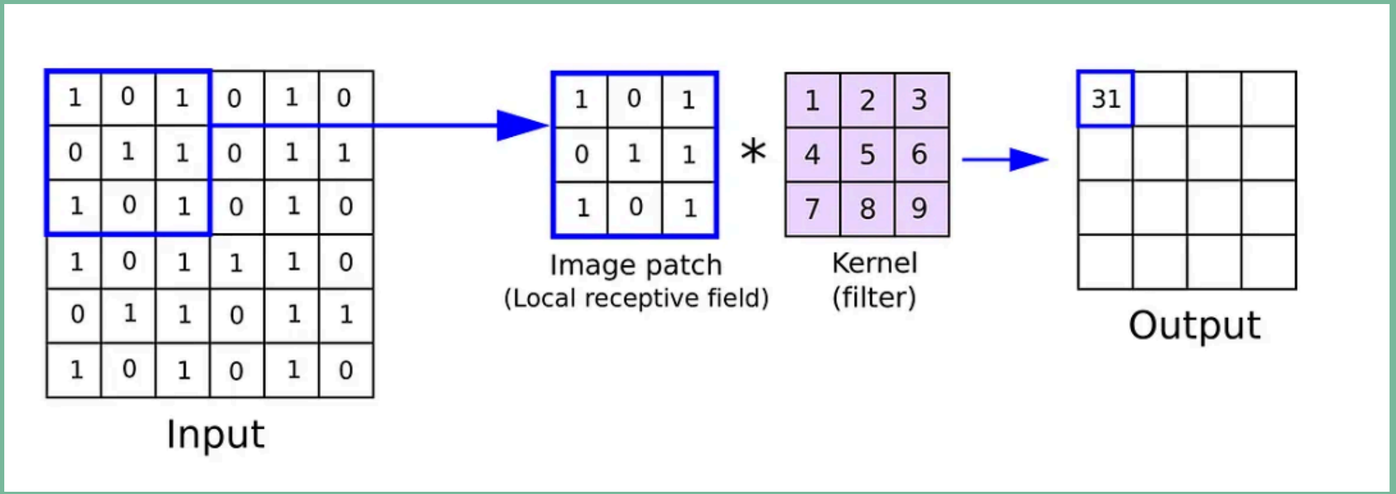


# CONVOLUTIONAL LAYER

The convolutional layer is the core building block of a CNN.
It applies small matrices called **kernels** (**filters**) to the input data in order to extract important features.
Each filter slides across the input and performs element-wise multiplication, producing a feature map that highlights specific patterns such as edges, textures, or local dependencies.

Output size (without padding):
Output size = (i−k)+1
Where:

- i = input size
- k = kernel size

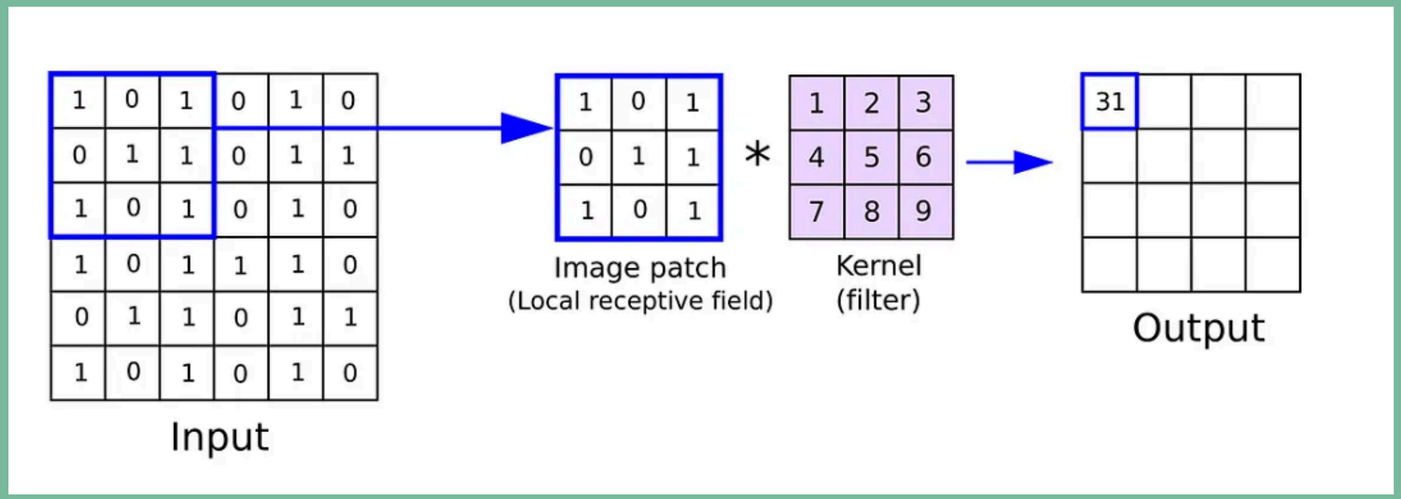


# KERNEL (FILTER)

A kernel (or filter) is a small matrix that acts as a feature detector.
Different kernels learn different patterns, for example:

- Edges
- Shapes
- Local correlations

During training, kernel values are automatically learned through backpropagation.



# STRIDE

Stride controls how far the kernel moves at each step during convolution.

- Stride = 1 → filter moves one unit at a time
- Stride > 1 → skips positions, reducing output size

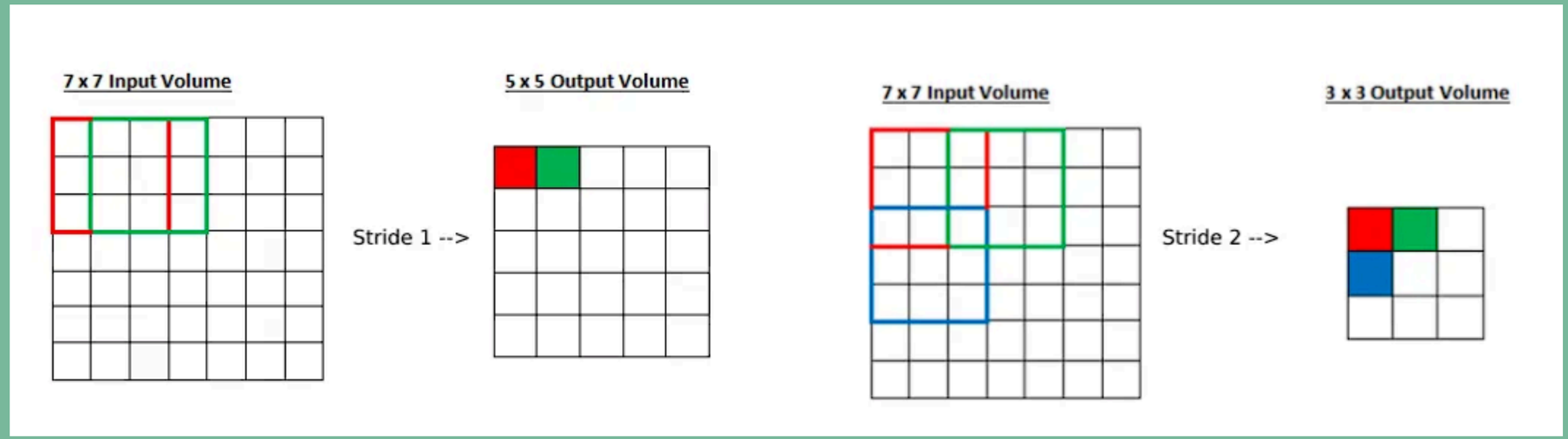
Output size with stride:

Output size = i-ks+1,

Where:

- s = stride size

Using a larger stride reduces computation but may lose fine-grained information.



# PADDING

Padding adds extra pixels (usually zeros) around the border of the input.

Purpose of padding:

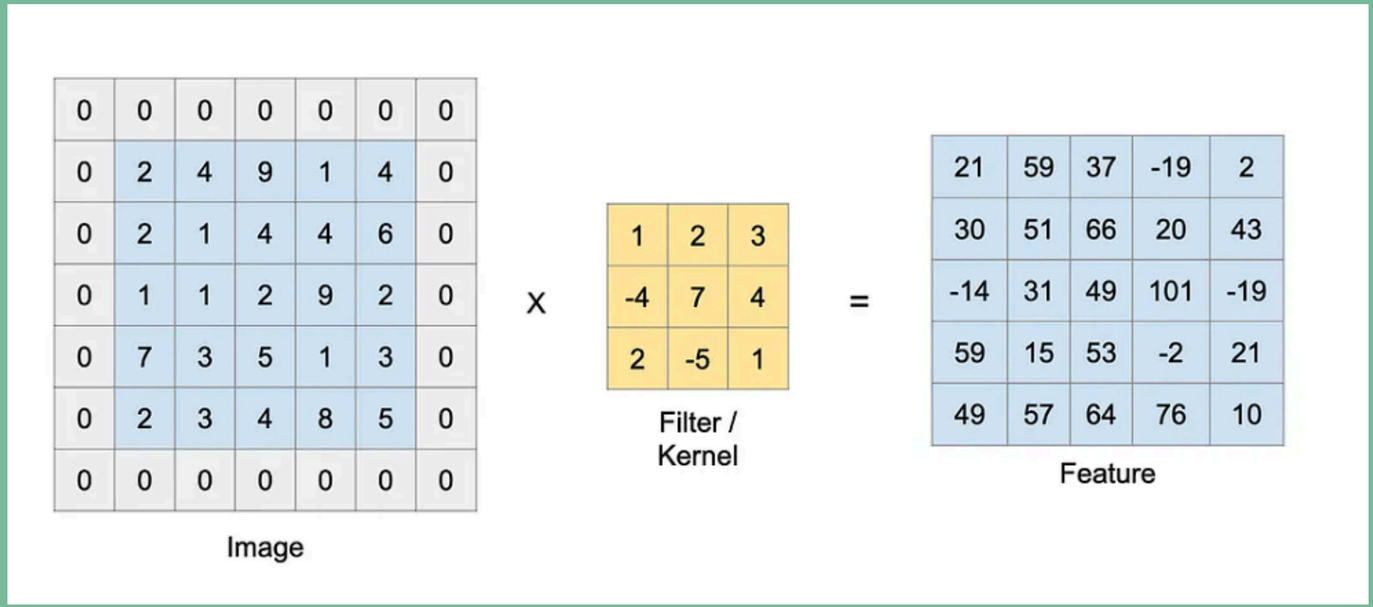
- Preserve spatial dimensions
- Prevent excessive shrinking of feature maps
- Retain low-level features at image boundaries

Output size with padding:

Output size = ((i-k+2p)/s)+1

Where:

- p = padding size



# POOLING LAYER

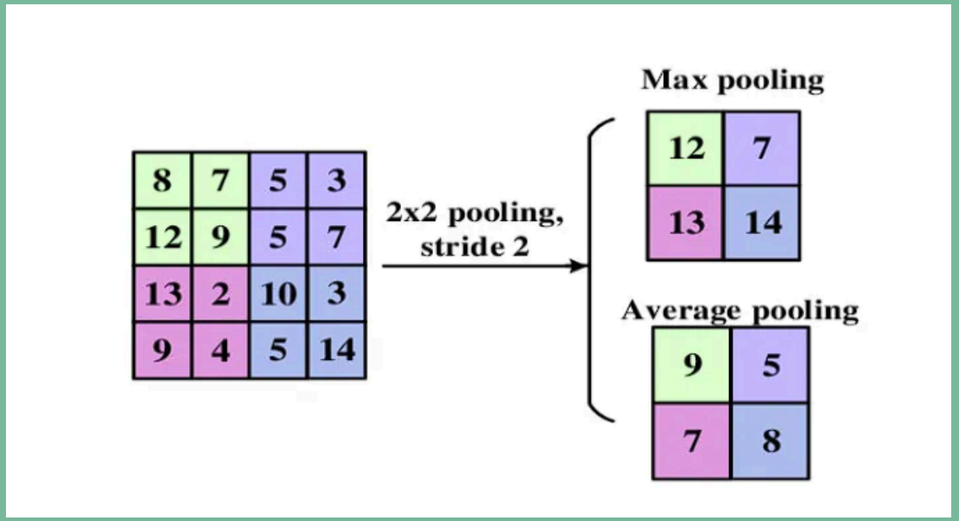
The pooling layer reduces the spatial dimensions of feature maps while retaining the most important information.

Main benefits:

- Reduces computational cost
- Controls overfitting
- Makes features more translation-invariant

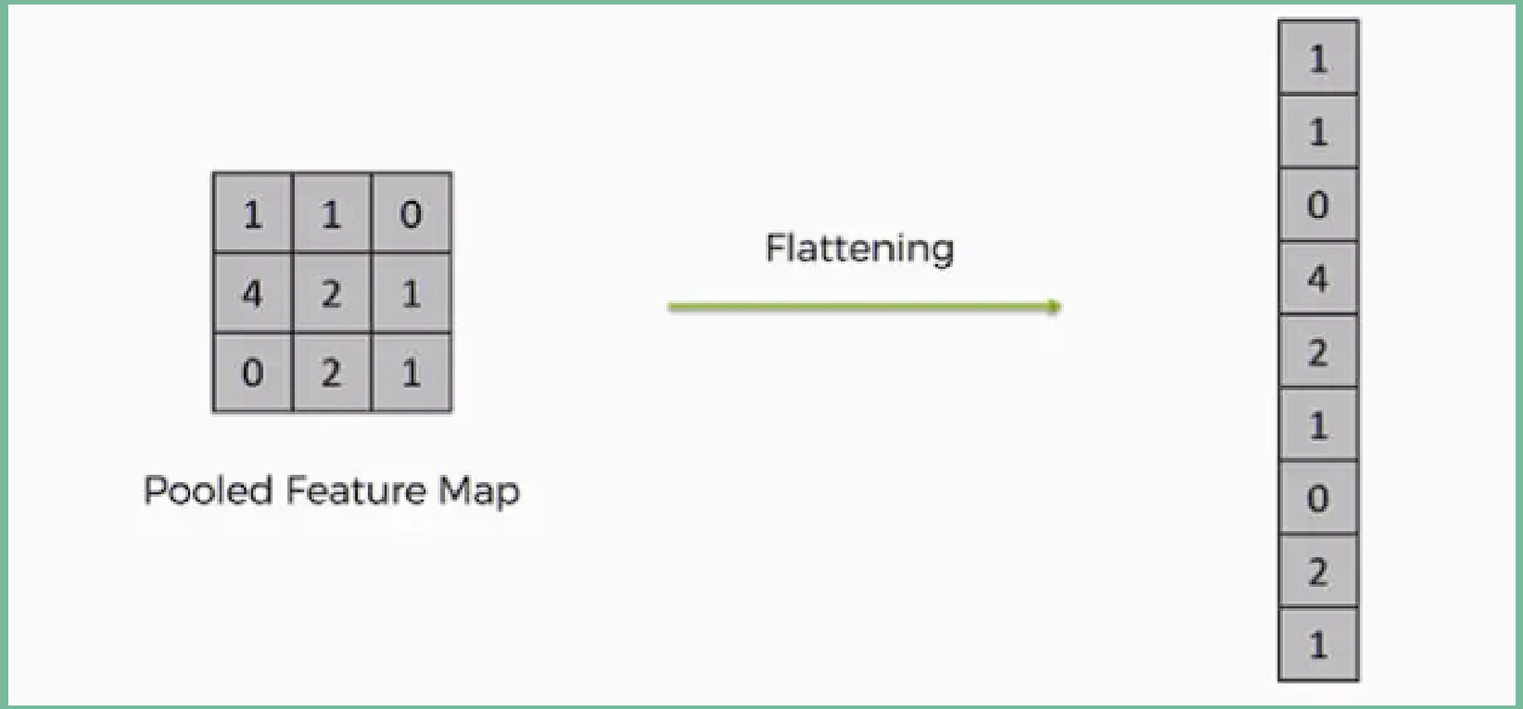
Common pooling types:

- Max Pooling → selects maximum value
- Average Pooling → computes average value



# FLATTEN LAYER

After convolution and pooling, the resulting feature maps are still multidimensional.
The flatten layer converts these feature maps into a one-dimensional vector, preparing the data for classification.
This vector becomes the input for the fully connected layers.



# FULLY CONNECTED (FC) LAYER

The fully connected layer connects every neuron from the previous layer to the next layer.
Its role:

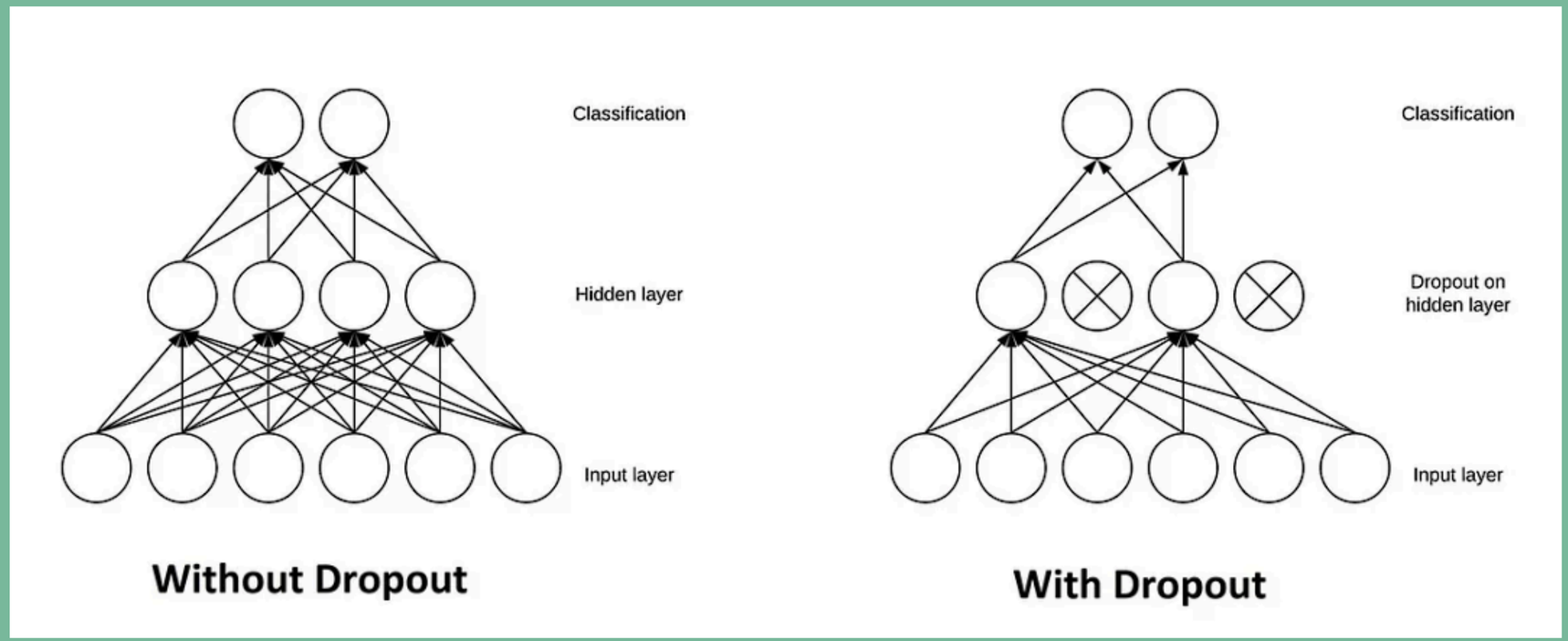
- Combine learned features
- Perform high-level reasoning
- Produce final predictions

FC layers typically appear near the end of the CNN architecture.

# DROPOUT LAYER

Dropout is a regularization technique used to reduce overfitting.
During training:

- Randomly disables a percentage of neurons
- Prevents dependency on specific neurons
- Improves generalization performance



# ACTIVATION FUNCTIONS

An Activation Function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction. There are several commonly used activation functions such as the ReLU, Softmax, tanH, and the Sigmoid functions. Each of these functions has a specific usage.

- Sigmoid : For a binary classification in the CNN model
- tanH : The tanh function is very similar to the sigmoid function. The only difference is that it is symmetric around the origin. The range of values, in this case, is from -1 to 1.
- Softmax : It is used in multinomial logistic regression and is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes.
- RelU : the main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.

# WHAT IS TF-IDF?

Kernel (Filter)

A kernel (or filter) is a small matrix that acts as a feature detector.

Different kernels learn different patterns, for example:

- Edges
- Shapes
- Local correlations

During training, kernel values are automatically learned through backpropagation.

# TERM FREQUENCY (TF)

It measure how often a word appears in a document

$$TF(t, d) = \frac{\text{Number of times term t appears in document d}}{\text{Total number of terms in document d}}$$

TF highlights local importance, “How important is this word inside this email?”

# INVERSE DOCUMENT FREQUENCY (IDF)

Measures how unique a word is across the entire dataset.

$$IDF(t) = log\frac{N}{1+df}$$

Where:

- N = total number of documents
- df = number of documents containing term t

Words like “yang”, “dan”, “kami” appear everywhere → low IDF
Words like “hadiah gratis”, “transfer”, “klik link” → high IDF

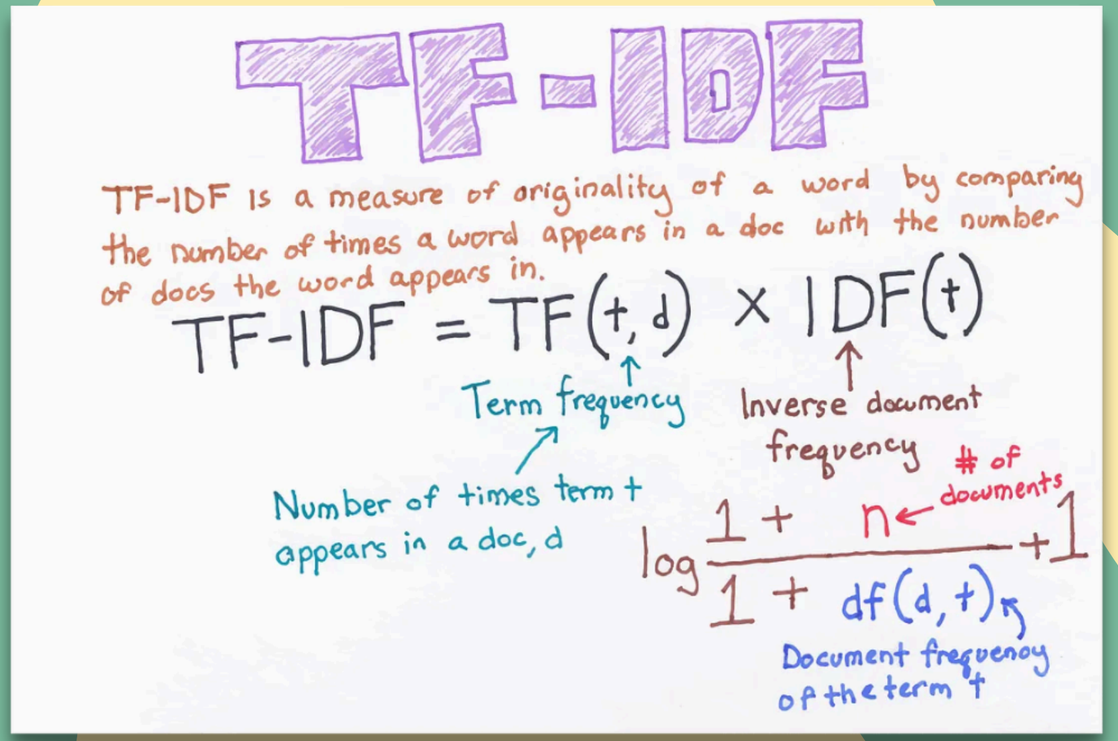
# TF-IDF AS COMBINED METRIC

**TF-IDF(T,D)=TF(T,D)×IDF(T)**

Meaning:

- High if the word appears frequently in this email but rarely across the corpus
- Low if the word is common everywhere (stopwords)

This helps the model focus on spam-related vocabulary.



# WHY TF-IDF

**Advantages**

- Extremely fast to compute
- Works well on short texts (emails, SMS)
- Captures keyword importance effectively
- Pairs very well with CNN, SVM, Logistic Regression, etc.
- Low computational cost → deployable on small systems

**Limitations**

- Cannot understand context
- Example:
  - “Transfer segera” (spam-like)
  - “Transfer ke folder lain” (not spam)
  - Same words but different meaning — TF-IDF cannot detect nuance.
- No semantic similarity
- Words like “promo”, “diskon”, “sale” are treated as unrelated.

# WHY TF-IDF WITH CNN?

CNN (Convolutional Neural Network) benefits from TF-IDF because:

CNN Strengths

- Detects local patterns such as n-grams:
  - “klik sekarang”
  - “hadiah menanti”
  - “promo terbatas”
- Learns nonlinear relationships
- Reduces risk of overfitting vs. fully connected networks

Result:

Even without word embeddings, TF-IDF + CNN can reach 59-60% accuracy, as seen in our project.

# WHAT IS INDOBERT?

IndoBERT is an Indonesian-language variant of Google's BERT model.

Key Characteristics:

- Built specifically on large Indonesian corpora
- Understands Indonesian grammar, slang, morphology, and sentence patterns
- Based on Bidirectional Transformers

IndoBERT is a deep contextual language model, meaning it understands:

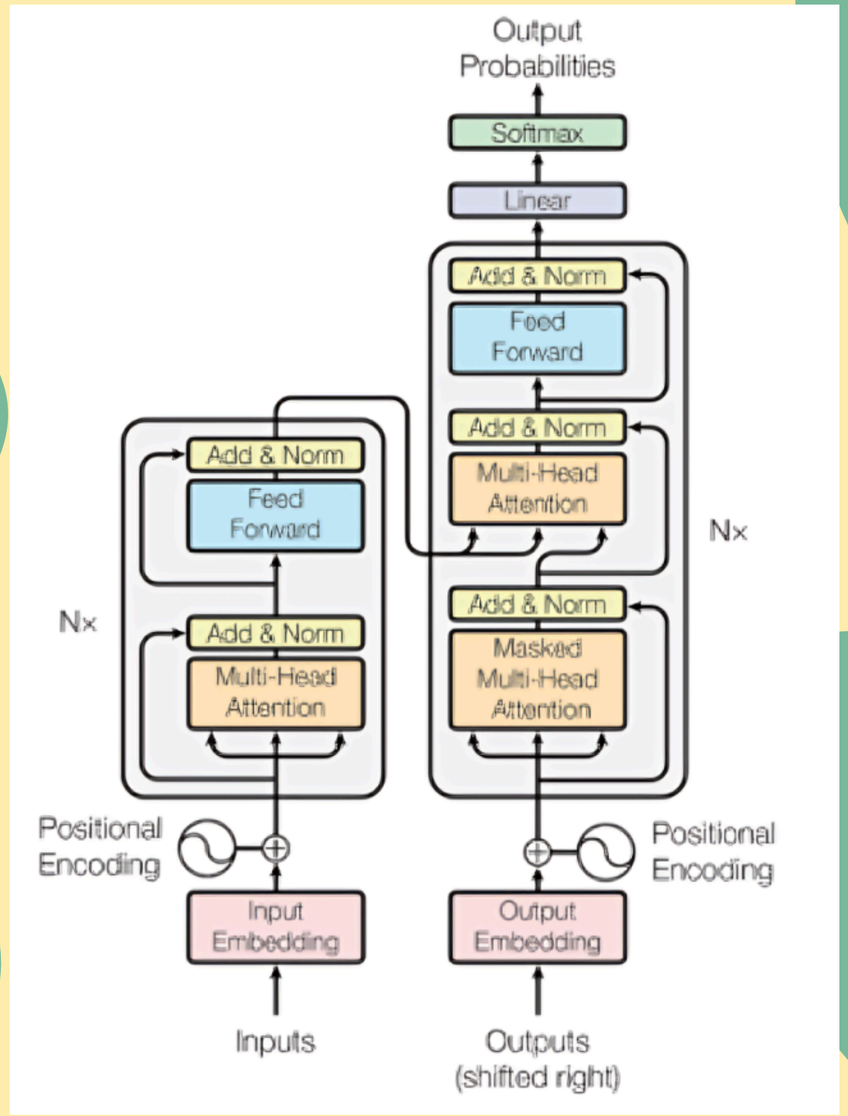
- Word meaning based on surrounding words
- Syntax (structure)
- Semantics (meaning)
- Intent & nuance

# TRANSFORMER ARCHITECTURE

IndoBERT is based on the Transformer encoder stack:

Key Components:

1. Self-Attention Mechanism
2. Allows the model to understand relationships between all words in a sentence simultaneously.
3. Example:
   - “Transfer uang Anda sekarang”
   - → strong spam context
4. Multi-Head Attention
5. Multiple attention heads learn different linguistic patterns.
6. Feed Forward Layer
7. Transforms attention output into contextual embeddings.
8. Layer Normalization + Residual Connections
9. Stabilizes training and preserves gradients.



# WHY INDOBERT > TF-IDF

1. **Contextual Understanding**

IndoBERT knows that:

- “Hadiah gratis untuk Anda” → spam
- “Hadiah ulang tahun untuk teman” → not spam

TF-IDF cannot make this distinction.

2. **Semantic Similarity**

IndoBERT knows:

- “Promo”, “diskon”, “sale”, “potongan harga”
- All have similar meaning.

3. **Handles Informal Indonesian**

IndoBERT understands:

- “klik link ini”
- “klik dong”
- “coba klik deh”

4. **Handles word order variation**

IndoBERT sees:

- “Segera klaim hadiah Anda”
- “Hadiah Anda segera klaim”

Same meaning, TF-IDF sees them as different.

# INDOBERT + CNN

Why add CNN on top of IndoBERT?

- IndoBERT outputs a sequence of contextual embeddings (128 tokens × 768 dimensions)
- CNN extracts local semantic features
- Global Max Pooling compresses strongest signals
- Dense layers produce final classification

This hybrid approach performs extremely well, especially for spam patterns.

# WHY INDOBERT IS SLOW

01 Requires GPU for efficient training

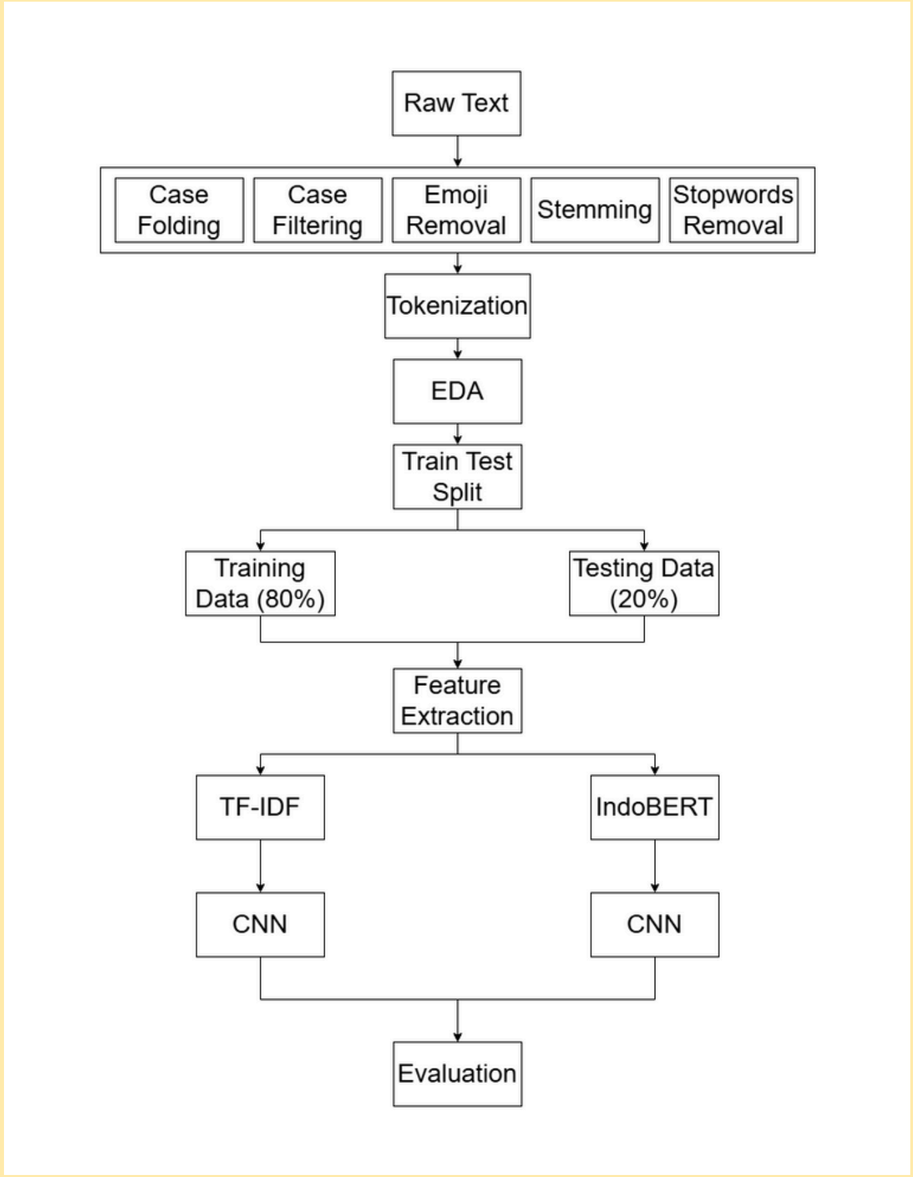
02 Large model size (hundreds of MB or even Gigabytes)

03 Slower Inference compared to TF-IDF

**But** The performance gain often justifies the cost. IndoBERT models are currently state of the art for Indonesian NLP tasks.

# METHODOLOGY

# ARCHITECTURE



# DATASET PREPROCESS

Filtering:

- Removed URLs, emails, emojis, symbols
- Removed retweets, advertisements, and duplicates

Text Normalization:

- Converted text to lowercase
- Removed non-alphabet characters
- Removed stopwords and short words

Stopwords Handling:

- Indonesian stopwords (Sastrawi)
- Additional custom stopwords (email terms, domains, company names)

Labeling:

- Encoded classes using LabelEncoder
- Labels mapped to numerical format

# DATASET PREPROCESS

```
STOPWORDS = set(StopWordRemoverFactory().get_stop_words())
STOPWORDS.update({
    "hou","kaminski","vince","enron","corp","edu","cc","re","fw",
    "subject","email","houston","pm","am","com","net","org",
    "ltd","co","inc","ect"
})

URL_RE = re.compile(r'https?://\S+|www\.\S+')
EMAIL_RE = re.compile(r'\S+@\S+')
NON_ALPHA = re.compile(r'[^a-zA-Z\s]')

if hasattr(emoji, "get_emoji_regexp"):
    EMOJI_RE = emoji.get_emoji_regexp()
    strip_emoji = lambda t: EMOJI_RE.sub(" ", t)
else:
    strip_emoji = lambda t: emoji.replace_emoji(t, replace=" ")

def clean_text(text: str) -> str:
    text = text.lower()
    text = URL_RE.sub(" ", text)
    text = EMAIL_RE.sub(" ", text)
    text = strip_emoji(text)
    text = NON_ALPHA.sub(" ", text)
```

# EXAMPLE

## SPAM

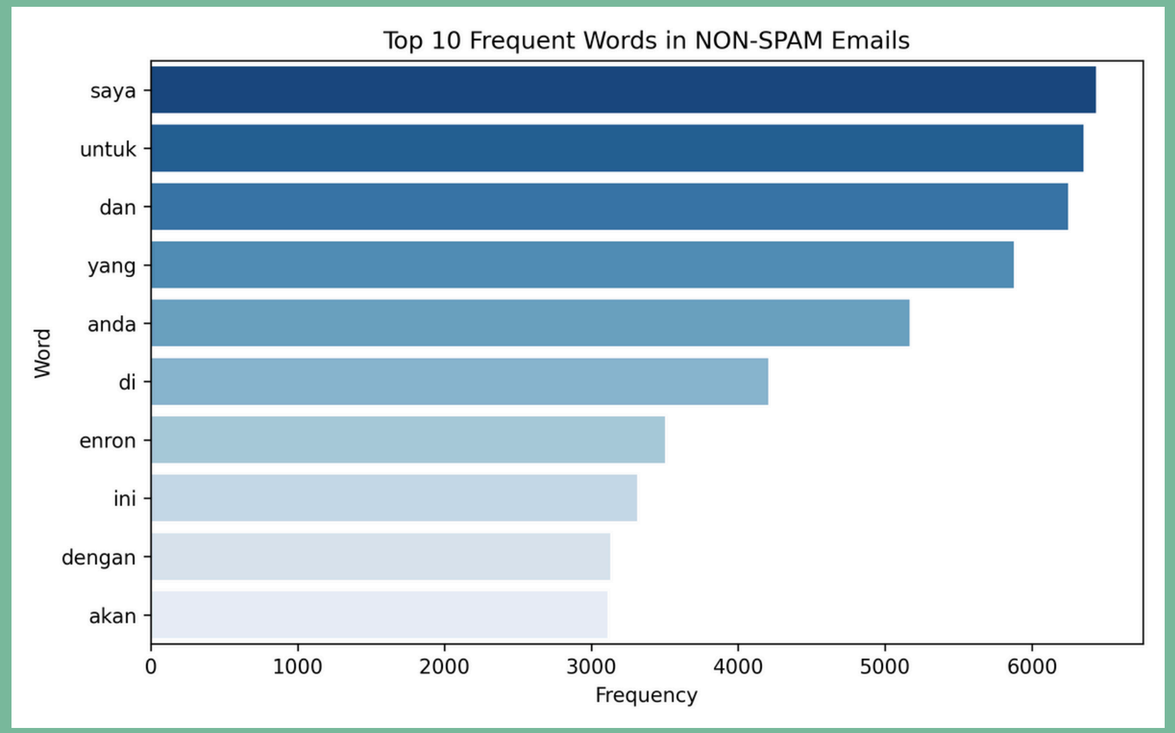
spam,"Secara alami tak tertahankan identitas perusahaan Anda sangat sulit u
yang luar biasa akan membuat tugas lebih mudah. Kami tidak berjanji bahwa H
dan tujuan praktis itu akan menjadi pasar saat ini; Tetapi kami berjanji ba
dilakukan untuk mencerminkan citra perusahaan Anda yang khas. Kenyamanan: L
Ketepatan: Anda akan melihat draft logo dalam tiga hari kerja. Keterjangkau
terbatas tanpa biaya tambahan bagi Anda untuk menjadi surethat Anda akan me

## NON-SPAM

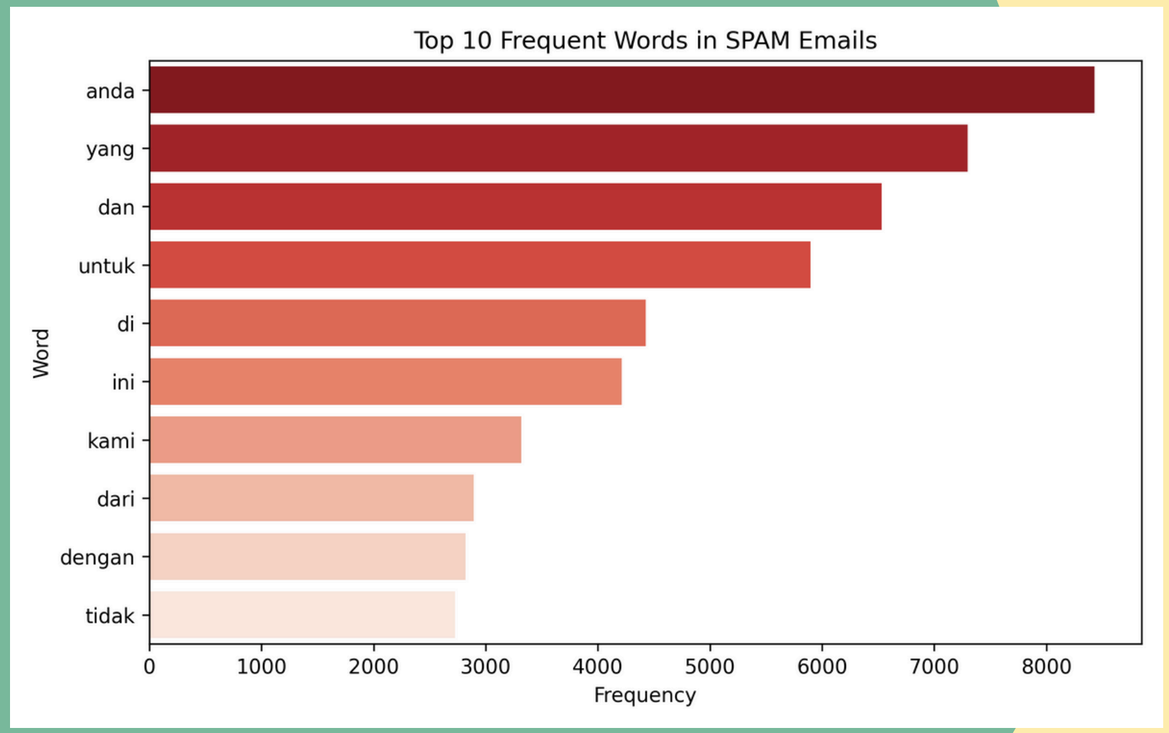
ham,"Peserta Seri Seminar Rice / Enron Finance Series / Enron Finance
/ ~ jgsfss /. Perhatikan bahwa ini adalah alamat web baru. Untuk keny
Charles Lee, Cornell 13 Okt George Allayannis, Darden 20 Okt William
yang berbaris untuk musim semi tetapi tanggal belum dijadwalkan: Tim
seperti yang telah kami lakukan di masa lalu, kami akan memposting ve
beranda speaker sehingga Anda dapat mengakses informasi biografinya.
(University of Houston), dan Vince Kaminski (Enron). Saya akan mengum
email ini. Silakan hubungi saya di Ostdiek @ Rice. EDU Jika Anda ingi
Keuangan Jones Graduate School of Management Rice University 6100 Mai

# WORD FREQ

## NON-SPAM




## SPAM



# MODEL TRAINING

Data Split:

- 80% Training – 20% Testing
- Optimizer:
- Adam (adaptive learning rate for faster convergence)

Loss Function:

- Sparse Categorical Cross-Entropy
- (suitable for multi-class text classification)

Training Strategy:

- Validation split: 10% from training data
- Batch size: 32
- Epochs: 20
- Early learning focus on feature extraction using CNN

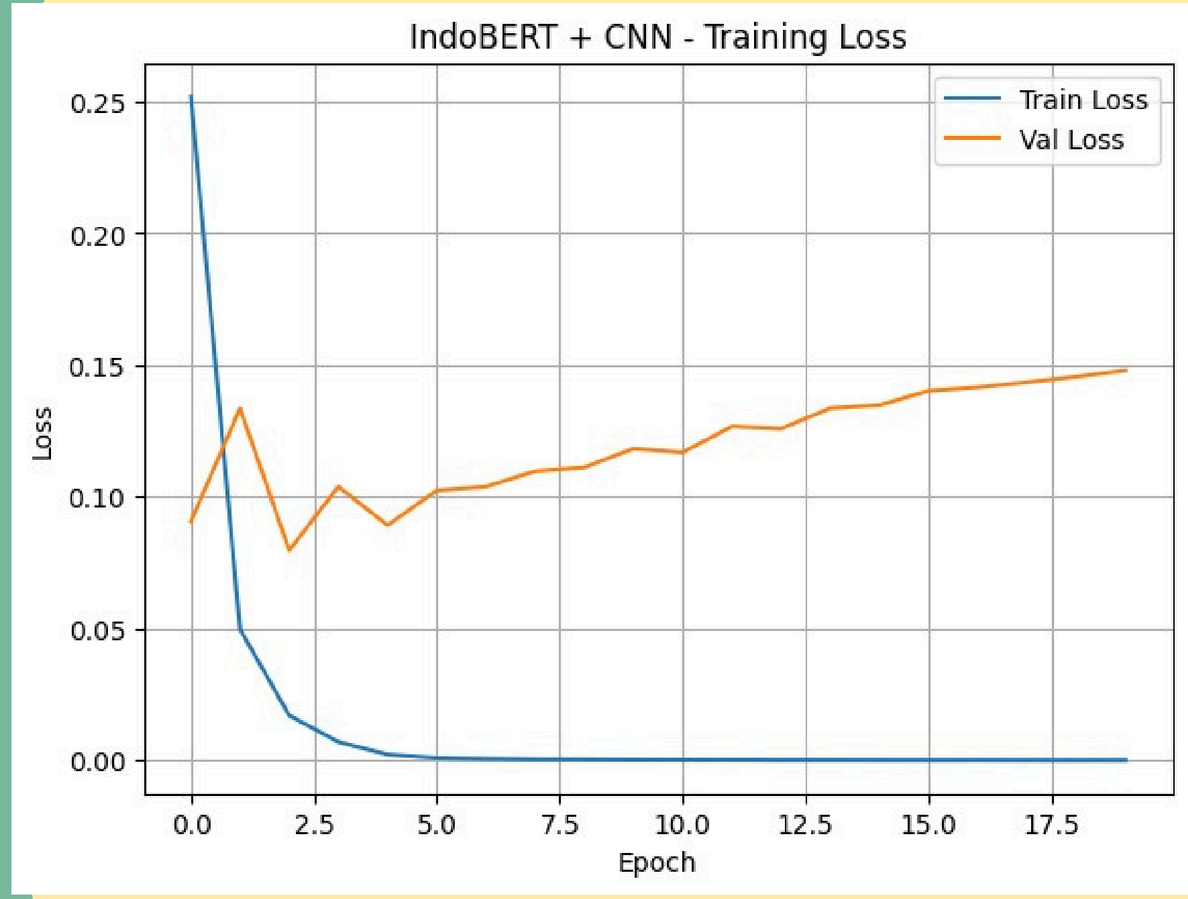
# MODEL TRAINING

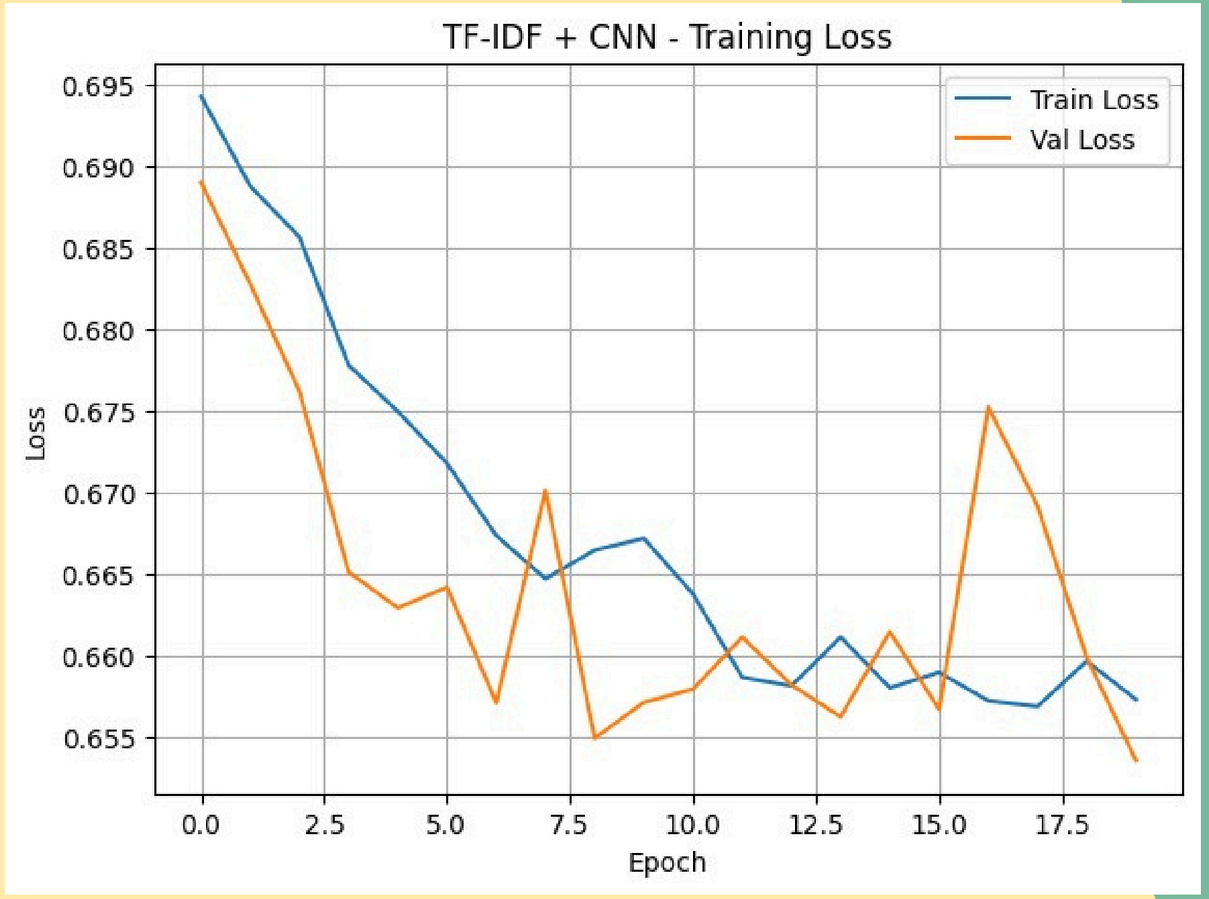
```python
# ============================================================
# BERT + CNN MODEL
# ============================================================
def build_bert_cnn(num_classes):
    # input tensors
    input_ids = layers.Input(shape=(MAX_LEN,), dtype=tf.int32, name="input_ids")
    attention_mask = layers.Input(shape=(MAX_LEN,), dtype=tf.int32, name="attention_mask")

    # backbone BERT
    bert_output = bert_backbone(
        input_ids=input_ids,
        attention_mask=attention_mask
    )[0]  # shape: (batch, MAX_LEN, 768)
```

```python
def fit_tfidf(texts):
    vectorizer = TfidfVectorizer(max_features=MAX_FEATURES)
    X = vectorizer.fit_transform(texts).toarray()  # shape: (n_samples, 5000)
    return vectorizer, X


# ============================================
# TRANSFORM TEXT → TF-IDF NUMERIC
# ============================================

def transform_tfidf(vectorizer, texts):
    X = vectorizer.transform(texts).toarray()
    return X


# ============================================
# BUILD CNN MODEL FOR TF-IDF INPUT
# ============================================

def build_tfidf_cnn(num_classes):
    model = Sequential()

    # Input shape: (5000,) → reshape menjadi (5000, 1)
    model.add(Conv1D(
        filters=64,
        kernel_size=5,
        activation='relu',
        input_shape=(MAX_FEATURES, 1)
    ))
```

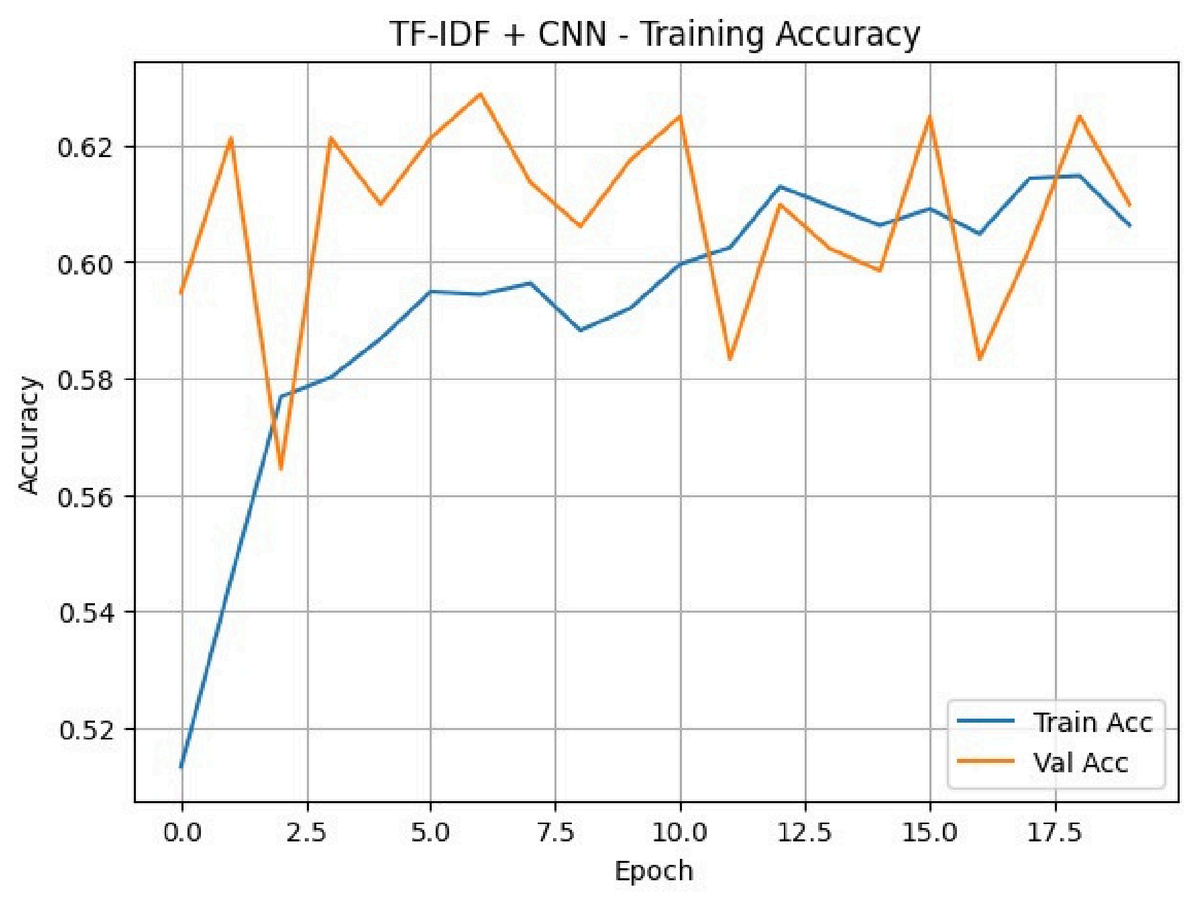
# RESULT

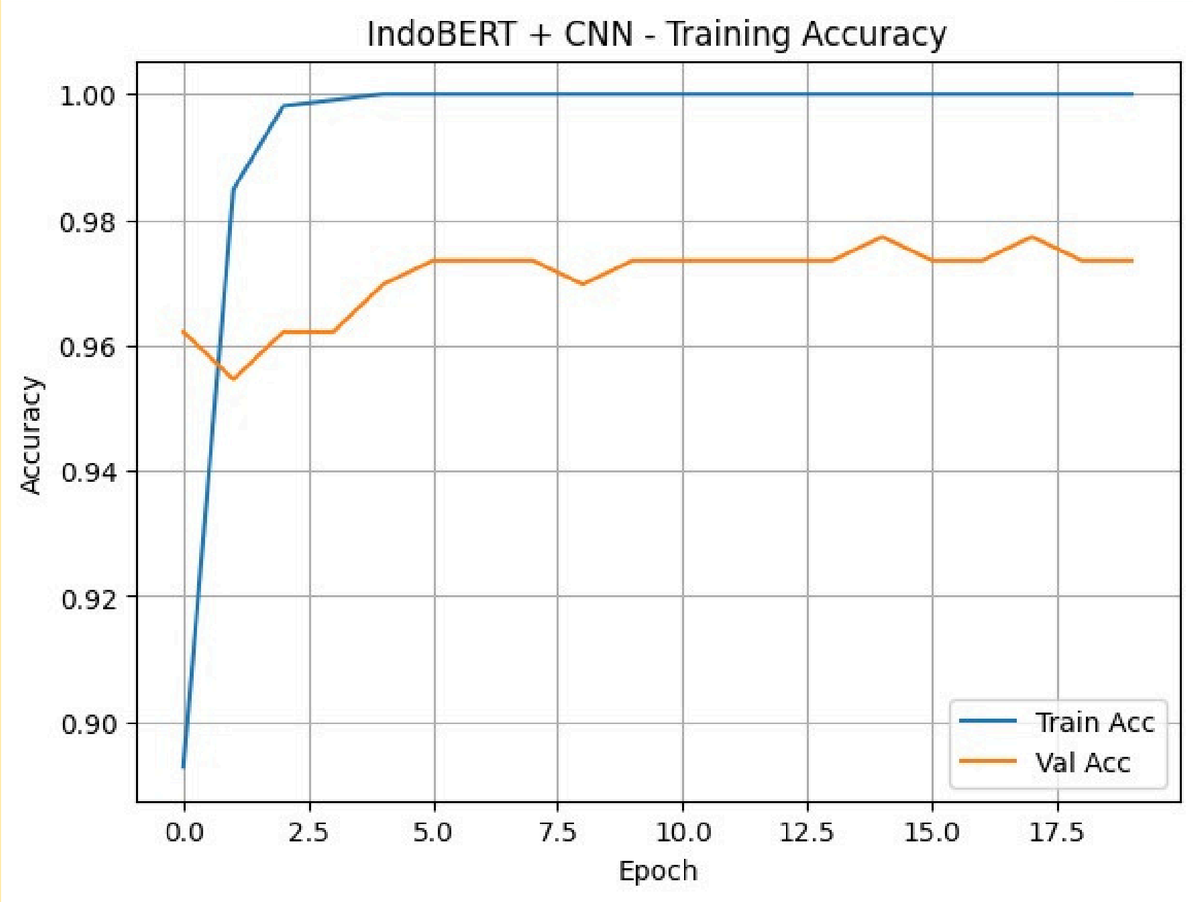
# TRAINING LOSS



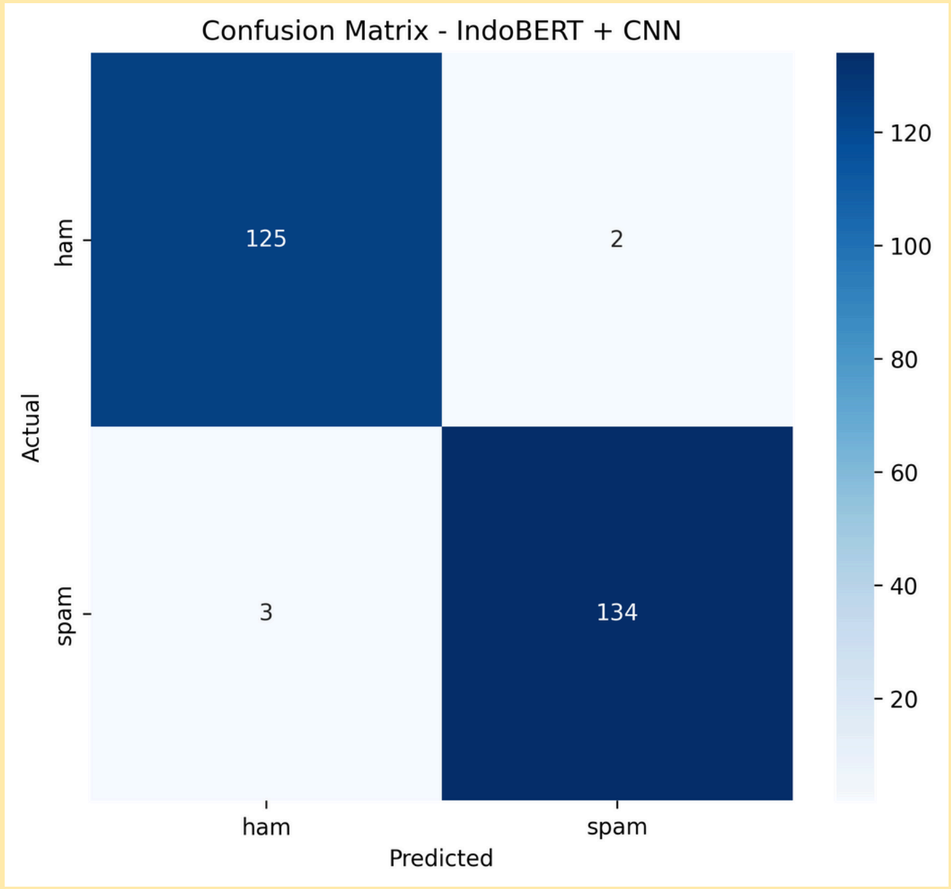
# TRAINING ACCURACY



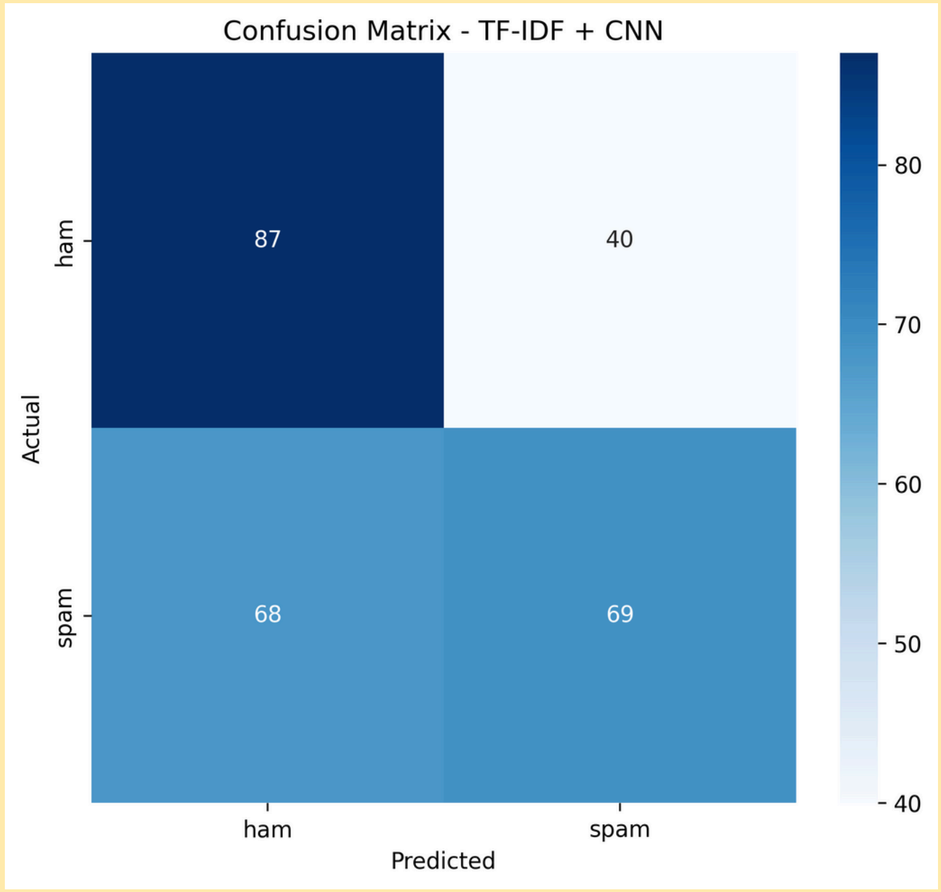



# HEATMAP

## IndoBERT + CNN




## TF-IDF + CNN



# TF-IDF

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 (Non-Spam) | 0.5612 | 0.685 | 0.617 | 127 |
| 1 (Spam) | 0.633 | 0.5036 | 0.5609 | 137 |
| **Accuracy** | **0.5909** | | | **264** |
| **Macro Avg** | 0.5971 | 0.5943 | 0.5889 | 264 |
| **Weighted Avg** | 0.5985 | 0.5905 | 0.5879 | 264 |

# INDOBERT

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 (Non-Spam) | 0.9765 | 0.9842 | 0.9803 | 127 |
| 1 (Spam) | 0.9852 | 0.9781 | 0.9816 | 137 |
| **Accuracy** | **0.981** | | | **264** |
| **Macro Avg** | 0.9809 | 0.9811 | 0.981 | 264 |
| **Weighted Avg** | 0.981 | 0.981 | 0.981 | 264 |

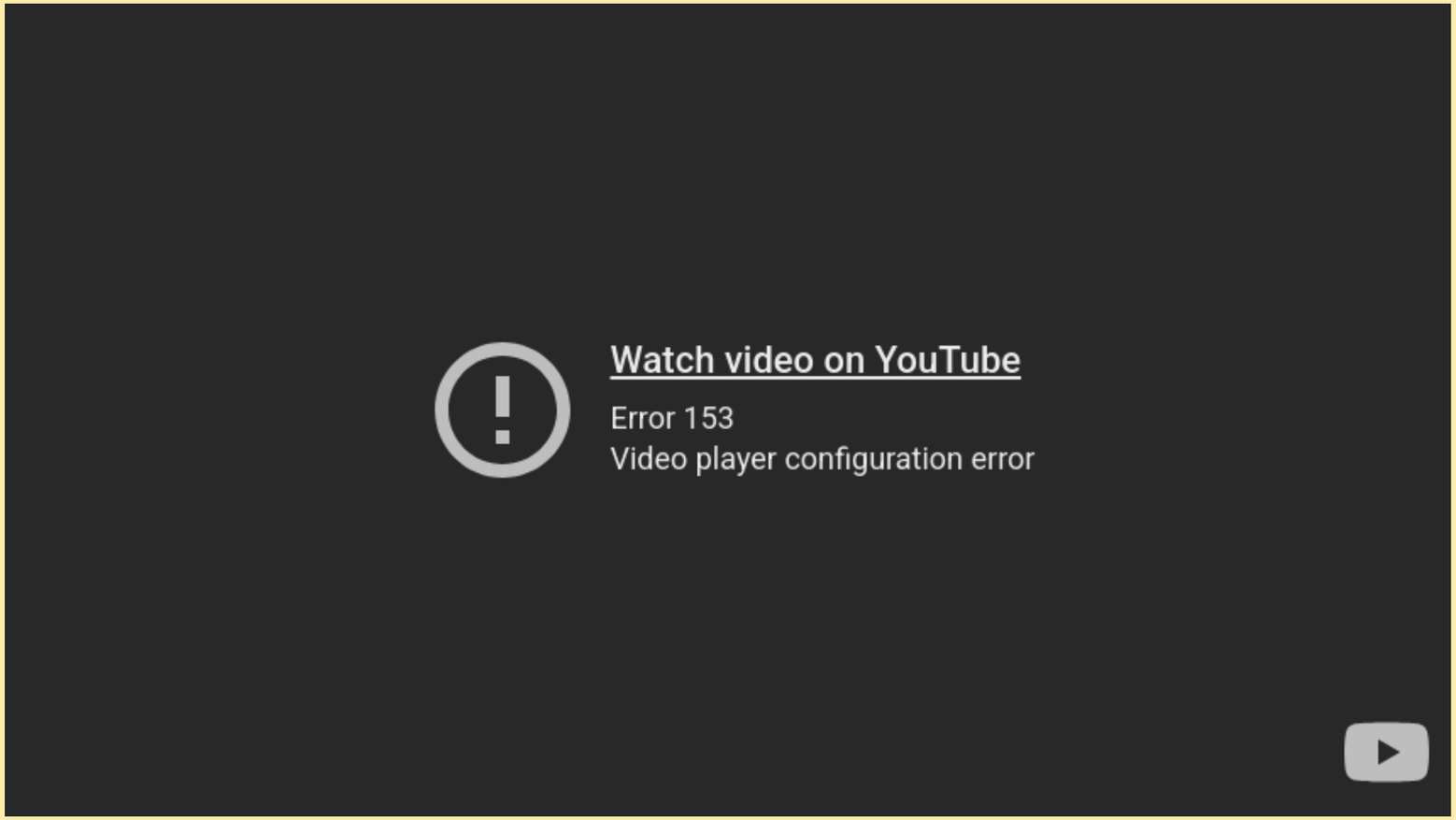
# TF-IDF

- Started with relatively high training loss
- (~0.69 at epoch 1)
- Training loss decreased slowly and remained above 0.65
- Validation accuracy fluctuated around 58% – 63%
- Final test accuracy reached ~59.8%
- Struggled to capture semantic meaning due to frequency-based representation

# INDOBERT

- Started with much lower training loss
- (~0.25 at epoch 1)
- Reached very low training loss
- (< 0.01 after a few epochs)
- Validation accuracy exceeded 96% from early epochs
- Final test accuracy achieved ~98.5%
- Converged faster and learned contextual Indonesian language features effectively

# Demo Video




https://youtu.be/uja_BXn9UUo

# CONCLUSION

Both models were able to learn patterns for Indonesian spam classification. TF-IDF + CNN worked as a baseline model but showed limited performance due to lack of contextual understanding. IndoBERT + CNN significantly outperformed TF-IDF + CNN, achieving much higher accuracy, precision, recall, and F1-score.

IndoBERT is preferable because:

- Lower training loss
- Better understanding of Indonesian linguistic context

Although slight overfitting was observed, IndoBERT maintained strong generalization performance.Overall, IndoBERT + CNN is more suitable for real-world Indonesian spam detection tasks, while TF-IDF + CNN can be used as a lightweight baseline.

# EXPERIMENT

# WHAT IS EMBEDDING

An embedding is a way to represent words as dense numerical vectors of fixed size.
Instead of treating words as isolated tokens, embeddings capture their meaning, context, and semantic relationships.
**Example**:
“money” and “cash” → vectors close to each other in embedding space.

# WHY USE EMBEDDING

Limitations of Traditional TF-IDF:

- Produces sparse vectors (mostly zeros)
- Cannot understand word meaning
- Cannot capture word order
- Vocabulary is fixed (new words = ignored or zero)

Embeddings Solve These Problems:

- Dense, compact vectors
- Capture semantic similarity
- Sensitive to local context (works well with CNN/LSTM)
- Better generalization across variations in language

# HOW DOES EMBEDDING WORK

## 1. Convert Words to integer indices

ex:

| Word | Index |
|---|---|
| prize | 120 |
| transfer | 87 |
| account | 64 |

Each unique word in the vocabulary gets an integer ID.
This step does not yet capture meaning, it only provides a way to reference each word numerically.

## 2. Embedding Layer maps index to dense vector

ex:

| Word | Vector |
|---|---|
| prize | [ 0.34, -0.12, 0.88, ... ] |
| transfer | [-0.22,  0.55, 0.03, ... ] |
| account | [ 0.10, -0.33, 0.25, ... ] |

The embedding layer maps each index to a learned vector. These vectors represent how words relate to each other.
What the embedding layer learns:

- Words appearing in similar contexts → get similar vectors
- Frequently co-occurring words → closer in vector space
- Semantic relationships emerge naturally

# Metrics Comparison

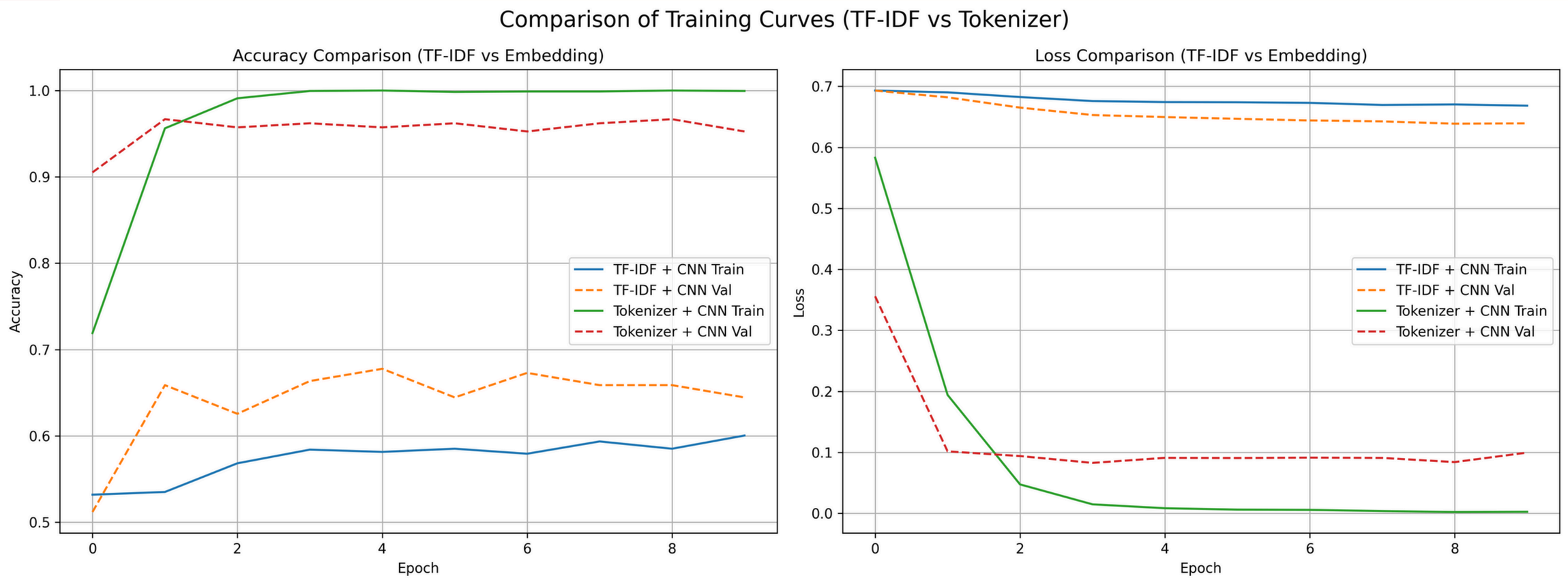
## TF-IDF

- Started with relatively high training loss
- (~0.69 at epoch 1)
- Training loss decreased slowly and remained above 0.65
- Validation accuracy fluctuated around 58% – 63%
- Final test accuracy reached ~59.8%
- Struggled to capture semantic meaning due to frequency-based representation

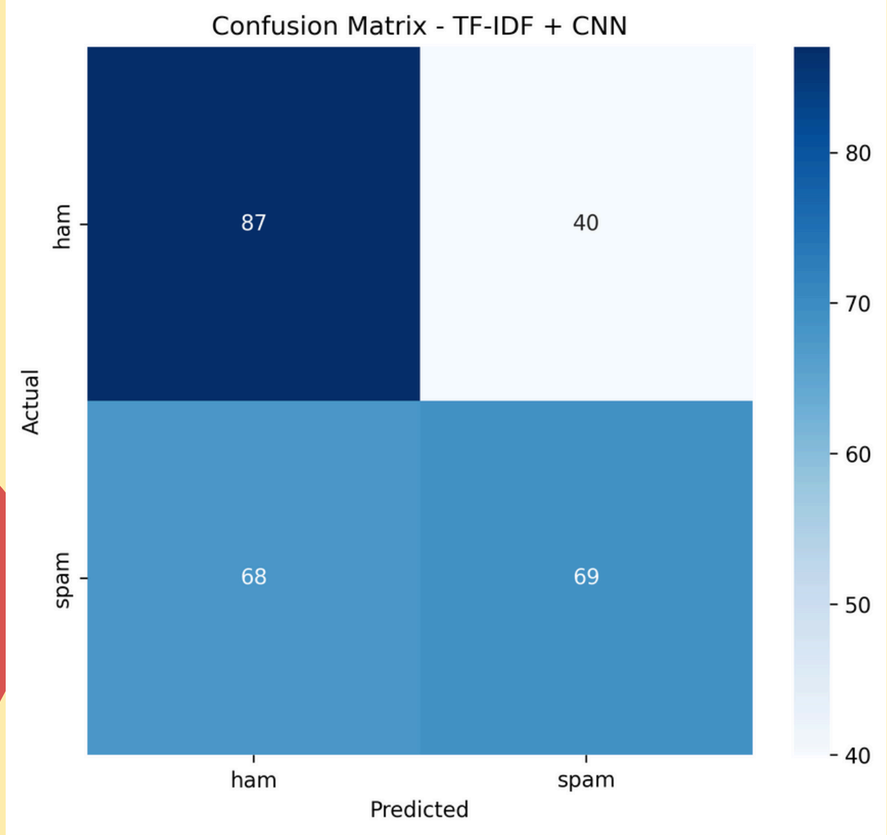
## Embedding

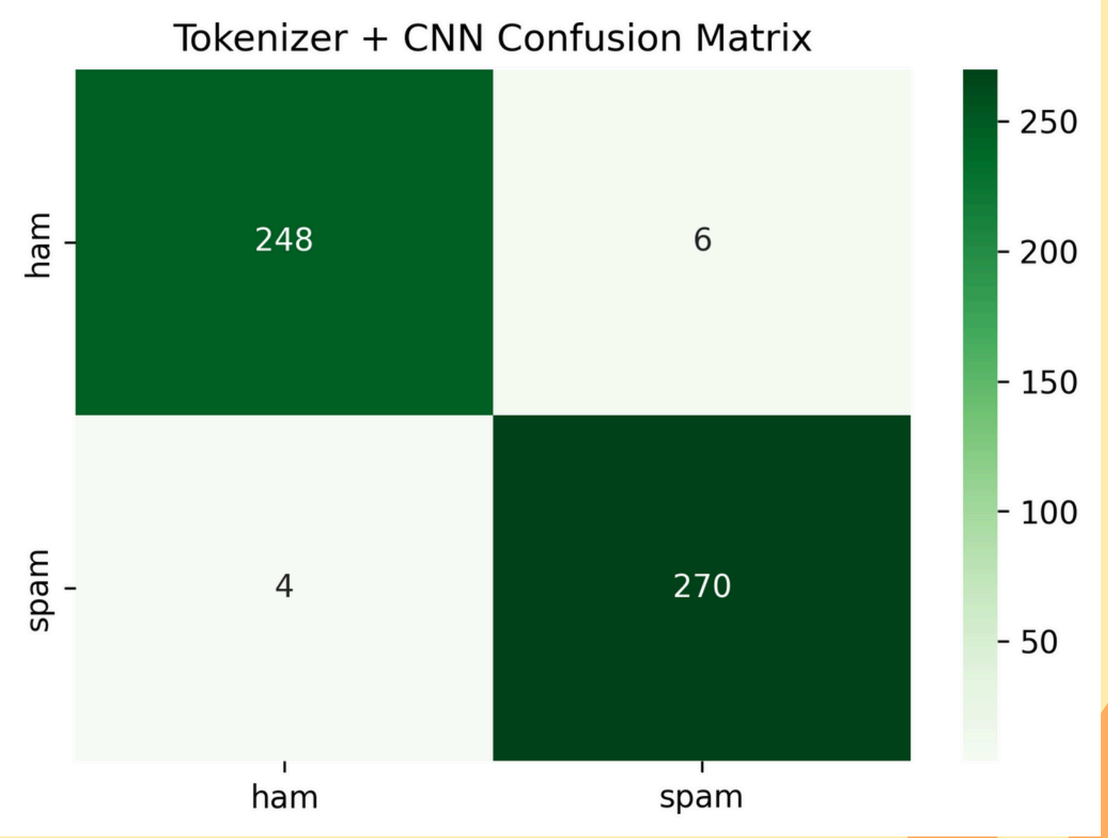
- Started with lower training loss
- (~0.58 at epoch 1)
- Reached very low training loss
- (< 0.02 after a few epochs)
- Validation accuracy exceeded 95% from early epochs
- Final test accuracy achieved ~98%
- Converged faster and learned contextual Indonesian language features effectively

# Metrics Comparison



# Metrics Comparison



# TF-IDF

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 (Non-Spam) | 0.5612 | 0.685 | 0.617 | 127 |
| 1 (Spam) | 0.633 | 0.5036 | 0.5609 | 137 |
| **Accuracy** | **0.5909** | | | **264** |
| **Macro Avg** | 0.5971 | 0.5943 | 0.5889 | 264 |
| **Weighted Avg** | 0.5985 | 0.5905 | 0.5879 | 264 |

# Embedding

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 (Non-Spam) | 0.9841 | 0.9762 | 0.9802 | 254 |
| 1 (Spam) | 0.9783 | 0.9854 | 0.9818 | 274 |
| **Accuracy** | **0.9811** | | | **528** |
| **Macro Avg** | 0.9812 | 0.9809 | 0.981 | 528 |
| **Weighted Avg** | 0.9811 | 0.9811 | 0.9811 | 528 |

# WHY EMBEDDING > TFIDF

Unlike TF-IDF (which creates sparse vectors), embeddings are:

- Dense: all values contain meaningful information
- Low-dimensional: e.g., 32D or 64D instead of thousands
- Semantic: capture meaning instead of just word frequency

This makes embeddings excellent for:

- Deep learning models (CNN, RNN, Transformers)
- Detecting subtle patterns (e.g., fraud language)
- Understanding context in sentences

# REFLECTION

# ANDREW

Through this project, I learned how significantly text representation choices influence the performance of deep learning models in Indonesian spam classification. Both TF-IDF + CNN and IndoBERT + CNN were able to learn meaningful patterns from the dataset. Implementing TF-IDF + CNN helped me understand how classical feature extraction methods can still serve as a solid and computationally efficient baseline. However, I also observed its limitations, particularly in capturing semantic and contextual information within Indonesian text.

Working with IndoBERT + CNN provided deeper insight into the importance of contextual embeddings. The model consistently achieved higher accuracy, precision, recall, and F1-score, which reinforced my understanding that transformer-based representations are more effective for language tasks involving nuanced meaning and context. Although I noticed slight overfitting during training, the IndoBERT-based model demonstrated strong generalization on unseen data. This experience taught me that higher model complexity does not necessarily reduce robustness when combined with appropriate training and evaluation strategies.

Beyond model development, this project also taught me the practical aspects of setting up a local deep learning environment. I learned how to properly configure a Conda environment, manage Python and library versions, and register kernels for use in tools such as VS Code and Jupyter Notebook. During development, I encountered several technical issues, including dependency conflicts, missing packages, and incorrect file or directory paths that caused the application or model to fail at runtime. Troubleshooting these problems required careful inspection of error messages, verification of directory structures, and consistent handling of relative and absolute paths when loading models, tokenizers, and vectorizers.

Through this debugging process, I developed a stronger understanding of how environment configuration and path management directly affect the reliability of a machine learning application. Resolving these issues improved my ability to diagnose errors systematically and ensured that the training pipeline, evaluation scripts, and deployed application were properly connected to the required resources.

Overall, this project helped me reflect on the trade-offs between model complexity, computational cost, and performance, while also strengthening my practical skills in environment setup and troubleshooting. While TF-IDF + CNN remains useful as a lightweight baseline, IndoBERT + CNN is more suitable for real-world Indonesian spam detection scenarios where accuracy and contextual understanding are crucial. This project strengthened my appreciation for selecting the right feature representation and building a stable development environment based on the problem domain and deployment constraints

# NICHOLAS WIRA ANGKASA

Working on this project has been a valuable learning experience that strengthened both my technical and analytical skills. Throughout the development of the TF-IDF + CNN and IndoBERT + CNN models, I gained a deeper understanding of how different text-representation techniques influence model performance, training behavior, and generalization.

One key lesson I learned is that model complexity does not always guarantee better results. TF-IDF, despite being a traditional method, performed quite well when combined with CNN, proving that classical techniques can still be highly effective when applied correctly. At the same time, implementing IndoBERT taught me the importance of leveraging pre-trained language models and understanding how they capture semantic relationships far beyond surface-level patterns.
During the project, I also improved my ability to troubleshoot issues, such as data-cleaning inconsistencies, incorrect vector shapes, model-loading problems, and Git version control conflicts. These challenges helped me become more disciplined in structuring code, documenting experiments, and organizing files to ensure reproducibility.

Another important reflection is the value of experiment tracking and visualization. By plotting confusion matrices, training curves, and comparing metrics side-by-side, I realized how crucial it is to evaluate models from multiple perspectives instead of relying solely on accuracy.

Finally, this project helped me grow not only as a developer but also as a learner. It taught me patience, problem-solving, and the importance of continuously iterating until the system works as expected. I now feel more confident in handling NLP workflows, deep learning pipelines, and experiment design, which I believe will be highly beneficial for my future academic and professional journey.

# THANK YOU