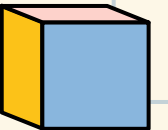
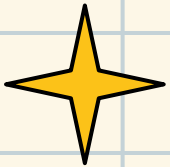
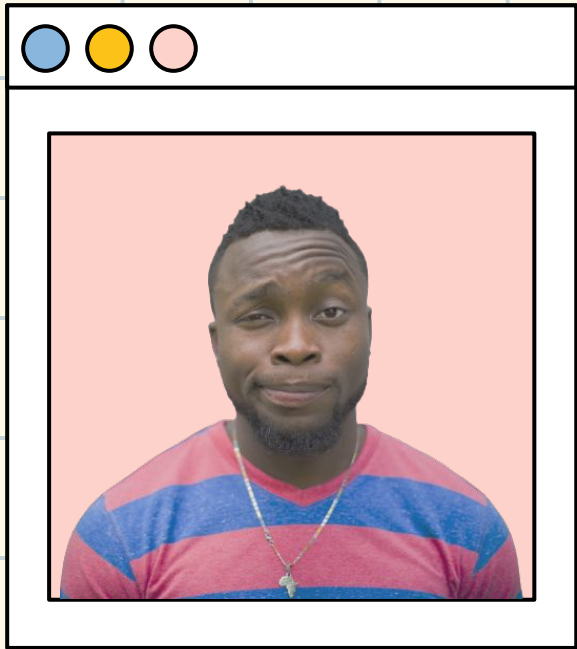


# Emotion Detection

x

with the use of facial expressions

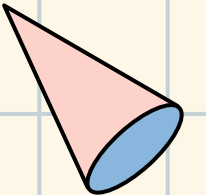
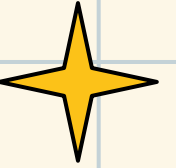




# Abstract

“Few realize how loud their expressions really are. Be kind with what you wordlessly say.”

— Richelle E. Goodrich

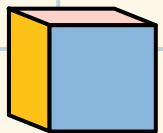
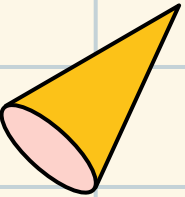




# Motivation



- Innovation and Technological Advancement
- Real-World Problem Solutions
- Commercial Potential
- Improving Interpersonal Relationships
- Scientific and Psychological Research
- Social and Ethical Responsibility



# Timeline



## 1980's

The beginnings of research in the field, with manual methods and human observations, without involving sophisticated technologies.

## 1990's

Initial attempts to use image processing technologies and pattern recognition algorithms for detecting and classifying facial emotions.

## 2000's

Increased use of machine learning algorithms, such as support vector machines and decision trees, but limitations in the complexity of facial expressions.

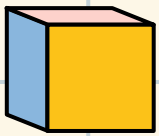
## 2010's

Advancement of deep neural networks, especially Convolutional Neural Networks (CNNs), leading to significantly improved accuracy in facial emotion classification.

## Present

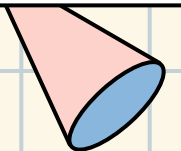
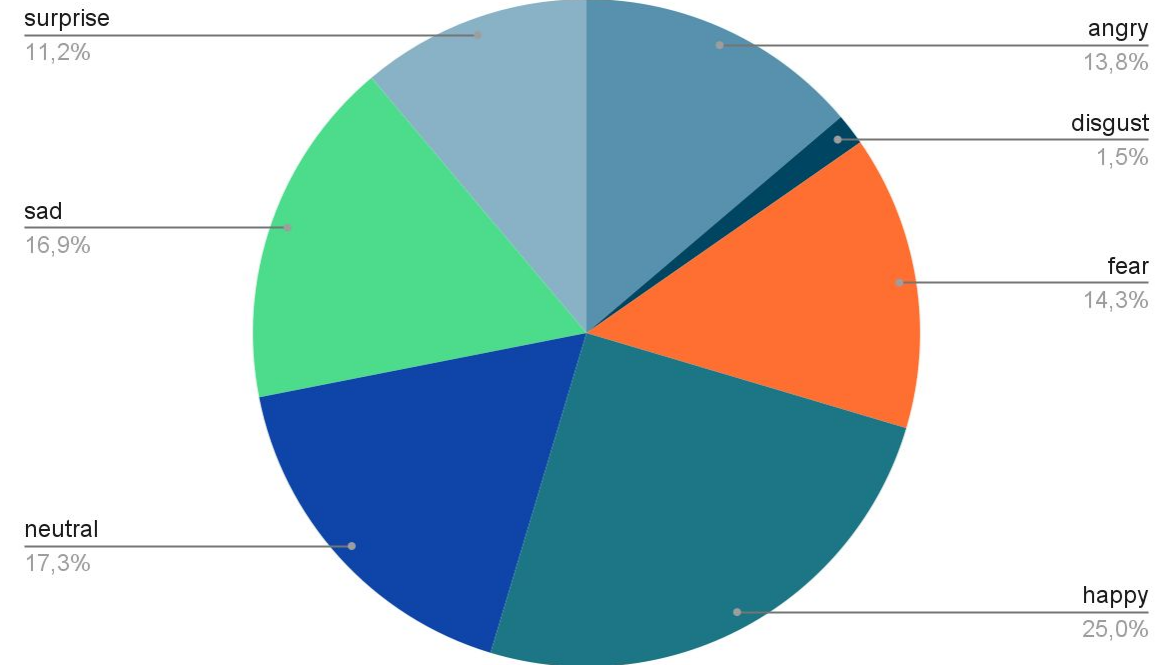
Continued progress through the use of more advanced neural network architectures, such as Long Short-Term Memory Networks (LSTMs) and Transformer Networks, and the development of multimodal learning technologies for integrating multiple data modalities.

X



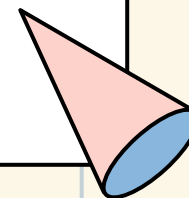
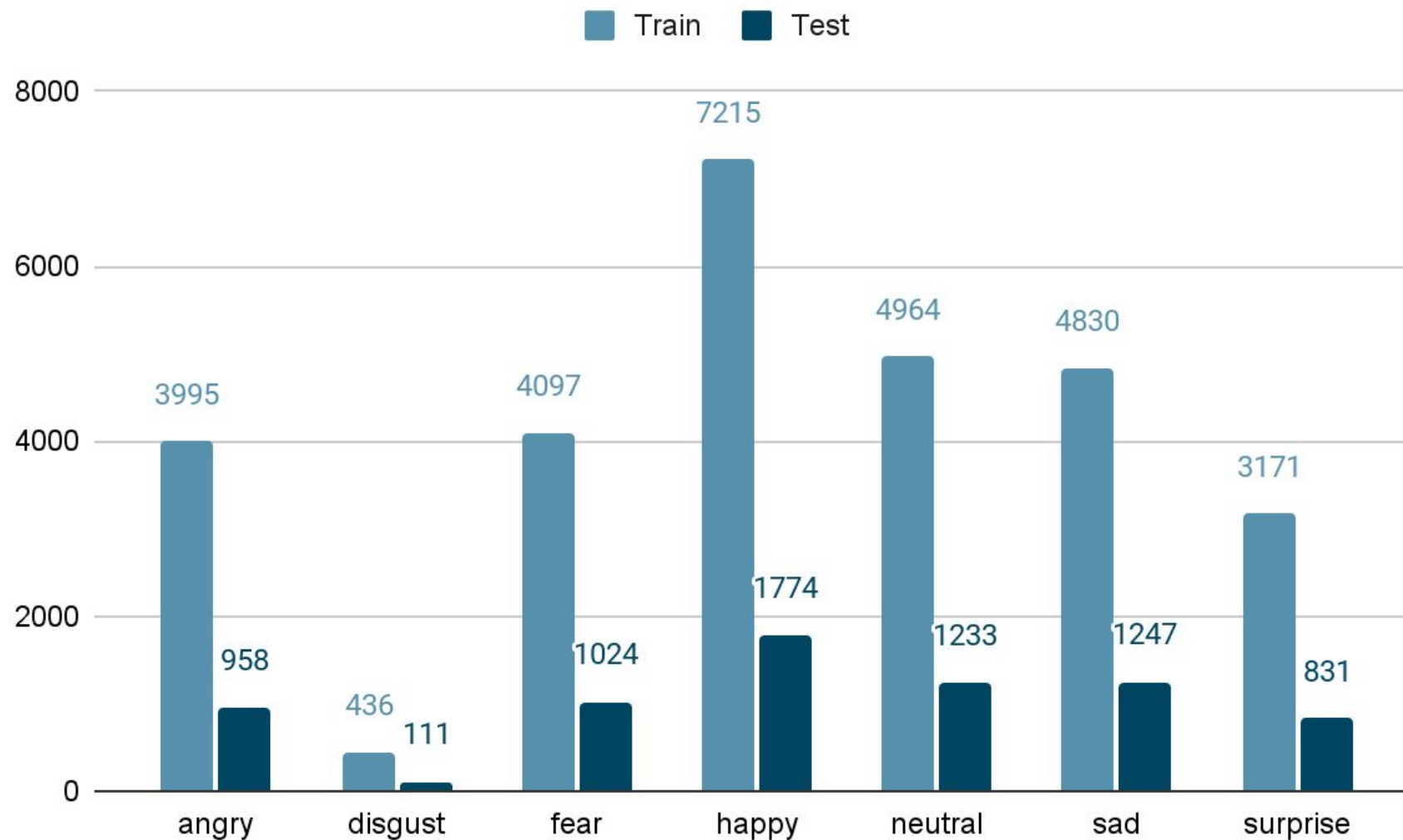
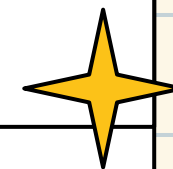
# Dataset

We will use the Facial Expression Recognition Challenge (FER2013) dataset that contains over 35,000 images labeled with 7 categories of emotions: anger, contempt, fear, happiness, sadness, surprise, and disgust.



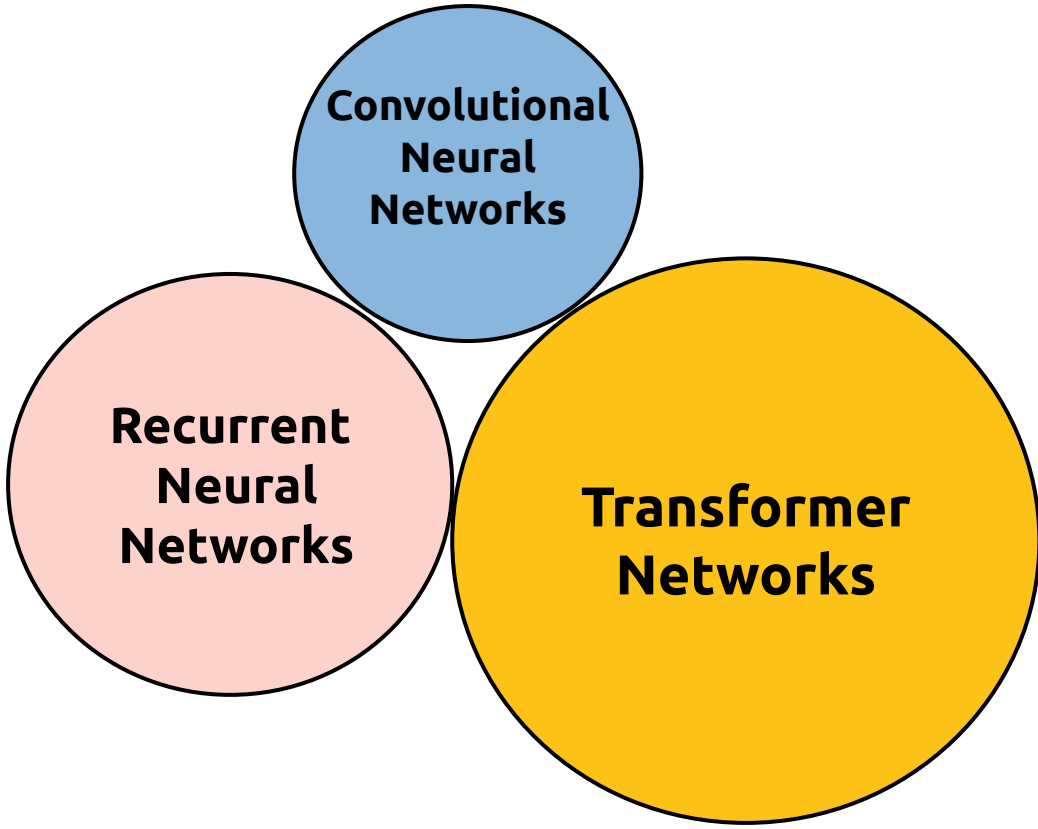


# More Numbers





# And what can we use?



**Convolutional  
Neural  
Networks**

**Recurrent  
Neural  
Networks**

**Transformer  
Networks**





# Convolutional Neural Networks

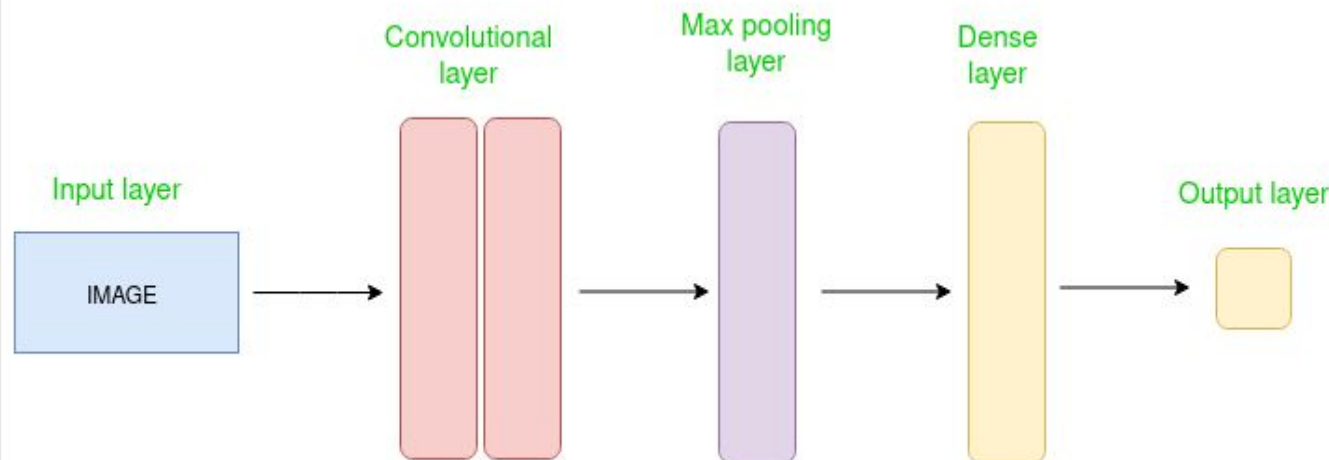
Convolutional Neural Networks (CNNs) are a type of artificial neural network designed specifically for processing structured grid data, such as images.

**Convolutional Layers:** These layers apply filters to input images, extracting features through convolution operations.

**Activation Function:** Non-linear functions like ReLU introduce non-linearity into the network, aiding in learning complex patterns.

**Pooling Layers:** These layers downsample feature maps, reducing spatial dimensions while retaining important information.

**Fully Connected Layers:** Flattened feature maps are passed through these layers, performing high-level reasoning and decision-making.



**Output Layer:** The final layer predicts class labels based on probabilities assigned to each class.

**Training:** CNNs are trained using backpropagation and gradient descent to minimize the difference between predicted and actual labels.

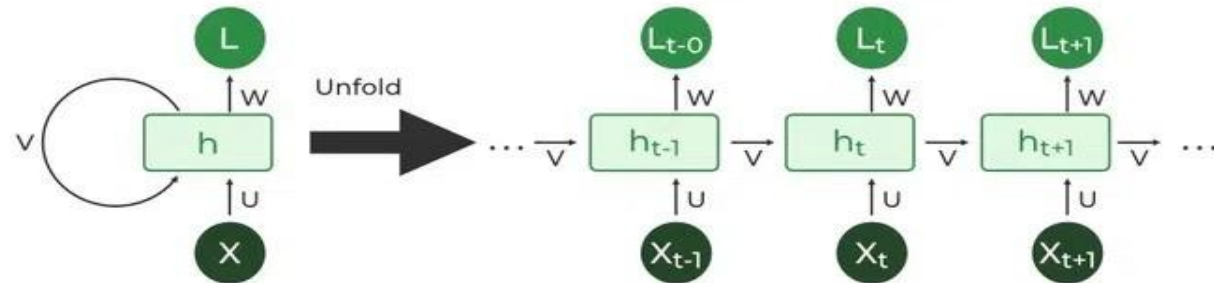




# Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a type of artificial neural network designed to process sequential data by maintaining an internal state or memory. Unlike traditional feedforward neural networks, which process each input independently, RNNs can capture dependencies and patterns across sequences of data.

**One to One :** This type of RNN behaves the same as any simple Neural network it is also known as Vanilla Neural Network. In this Neural network, there is only one input and one output.



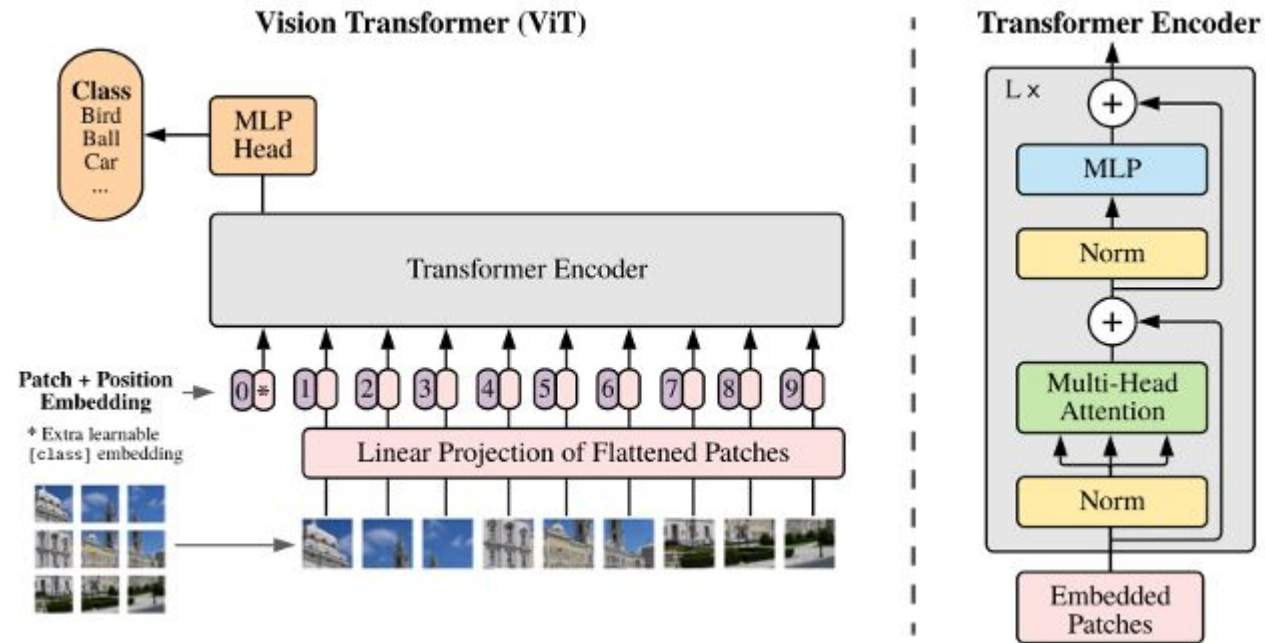
**One To Many :** In this type of RNN, there is one input and many outputs associated with it. One of the most used examples of this network is Image captioning where given an image we predict a sentence having Multiple words.

**Many to One :** In this type of network, Many inputs are fed to the network at several states of the network generating only one output. This type of network is used in the problems like sentimental analysis. Where we give multiple words as input and predict only the sentiment of the sentence as output.

**Many to Many :** In this type of neural network, there are multiple inputs and multiple outputs corresponding to a problem. One Example of this Problem will be language translation. In language translation, we provide multiple words from one language as input and predict multiple words from the second language as output.

# Transformer Networks

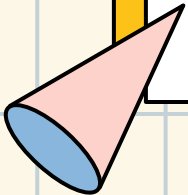
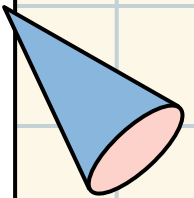
A more radical evolution in Neural Networks for Computer Vision, is the move towards using Vision Transformers (ViT) as a CNN-backbone replacement. Inspired by the astounding performance of Transformer models in Natural Language Processing (NLP), research has moved towards applying the same principles in Computer Vision. Notable examples, among many others, are XCiT, PiT, DeiT and SWIN-Transformers. Here, analogously to NLP processing, images are essentially treated as sequences of image patches, by modeling feature maps as vectors of tokens, each token representing an embedding of a specific image patch.





# ●●● But how do Vision Transformers work?

- **Image Patching:**
  - Break down the input image into a sequence of non-overlapping patches.
  - Each patch represents a local region of the image and typically contains a fixed number of pixels.
  - Allows the model to process the image sequentially, similar to how tokens are processed in NLP tasks.
- **Token Embedding:**
  - Linearly project each patch into a lower-dimensional embedding space, producing patch embeddings.
  - These embeddings serve as input tokens for subsequent transformer layers.
  - A learnable "class" token is appended to the sequence to represent global information about the entire image.
- **Transformer Encoder:**
  - Feed patch embeddings and the class token into a stack of transformer encoder layers.
  - Each transformer layer comprises multi-head self-attention mechanisms and feedforward neural networks.
  - Enables the model to capture both local and global relationships within the image.
  - Self-attention allows the model to attend to relevant patches and aggregate information across the entire image.
- **Classification Head:**
  - In image classification tasks, pass the output of the transformer encoder through a classification head.
  - The classification head consists of one or more fully connected layers followed by a softmax function.
  - Predicts the probability distribution over the output classes, enabling the model to classify the input image into different categories.
- **Task-Specific Heads:**
  - For other computer vision tasks like object detection or semantic segmentation, attach additional task-specific heads to the transformer encoder output.
  - These heads may include convolutional layers, spatial attention mechanisms, or other specialized architectures tailored to specific task requirements.

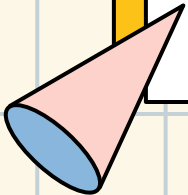
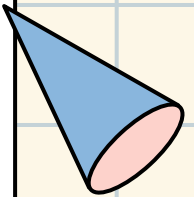




# Pros and Cons



	CNNs	RNNs	Transformer Networks
pros	<b>Feature Learning</b> <b>Translation Invariance</b> <b>Parameter Sharing</b> <b>Hierarchical Structure</b> <b>Effective Training</b>	<b>Temporal Modeling</b> <b>Variable-Length Inputs</b> <b>Contextual Understanding</b> <b>Transfer Learning</b> <b>Attention Mechanisms</b>	<b>Global Contextual Understanding</b> <b>Parallelization</b> <b>Scalability</b> <b>Attention Mechanism</b> <b>Transfer Learning</b>
CONS	<b>Large Data Requirements</b> <b>Computational Resources</b> <b>Overfitting</b> <b>Interpretability</b> <b>Limited Contextual Understanding</b>	<b>Vanishing Gradient</b> <b>Sequential Processing</b> <b>Short-Term Memory</b> <b>Difficulty in Capturing Spatial Information</b> <b>Training Instability</b>	<b>High Memory Requirements</b> <b>Limited Spatial Understanding</b> <b>Data Efficiency</b> <b>Training Instability</b> <b>Interpretability</b>





# Cu ce am lucrat?

## **CNN**

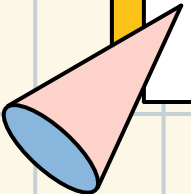
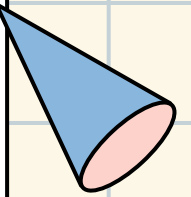
- ResNet101
- VGG16
- Custom CNN

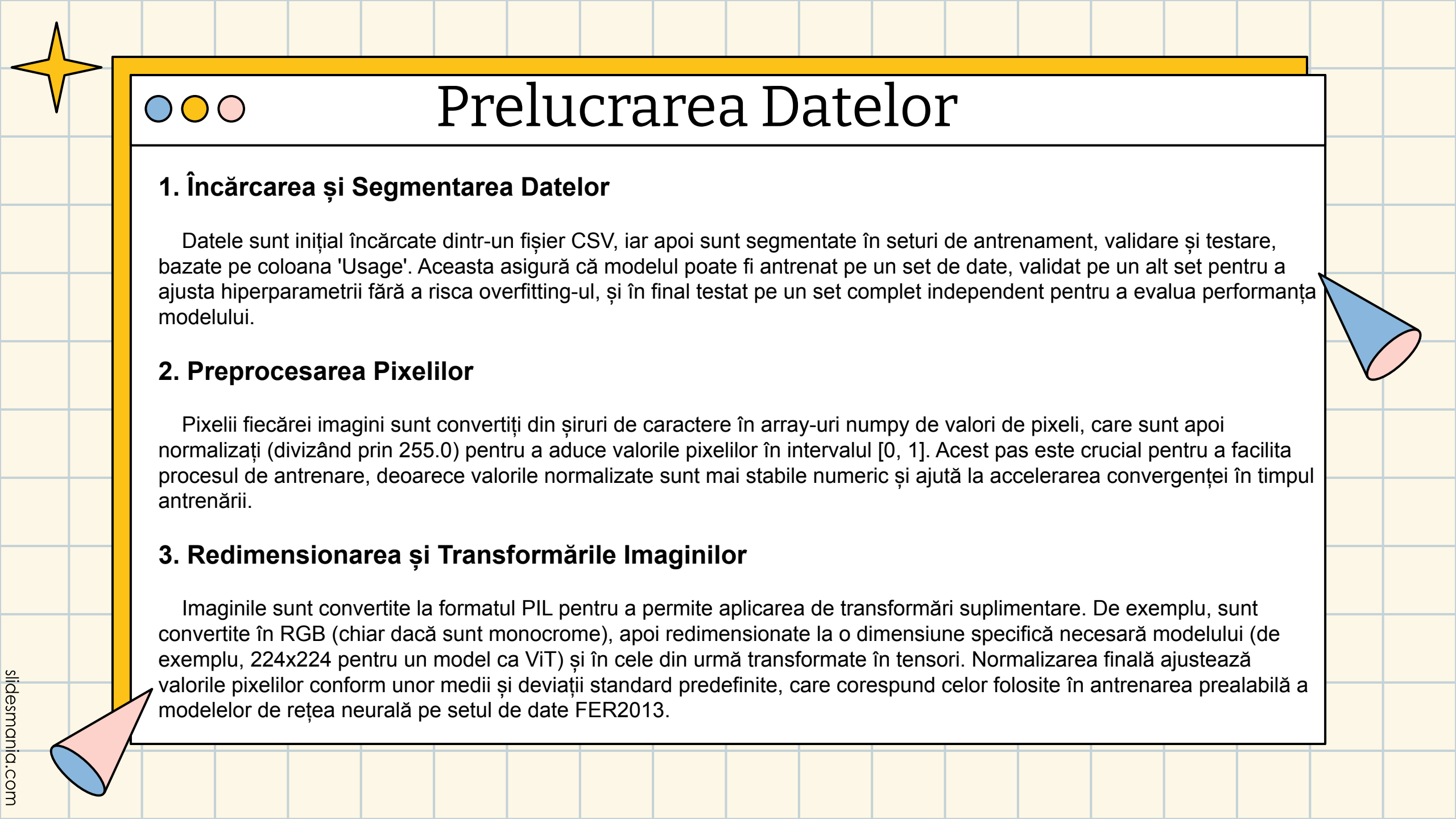
## **RNN**

- Custom RNN

## **ViT model**

- Custom ViT





# Prelucrarea Datelor

## 1. Încărcarea și Segmentarea Datelor

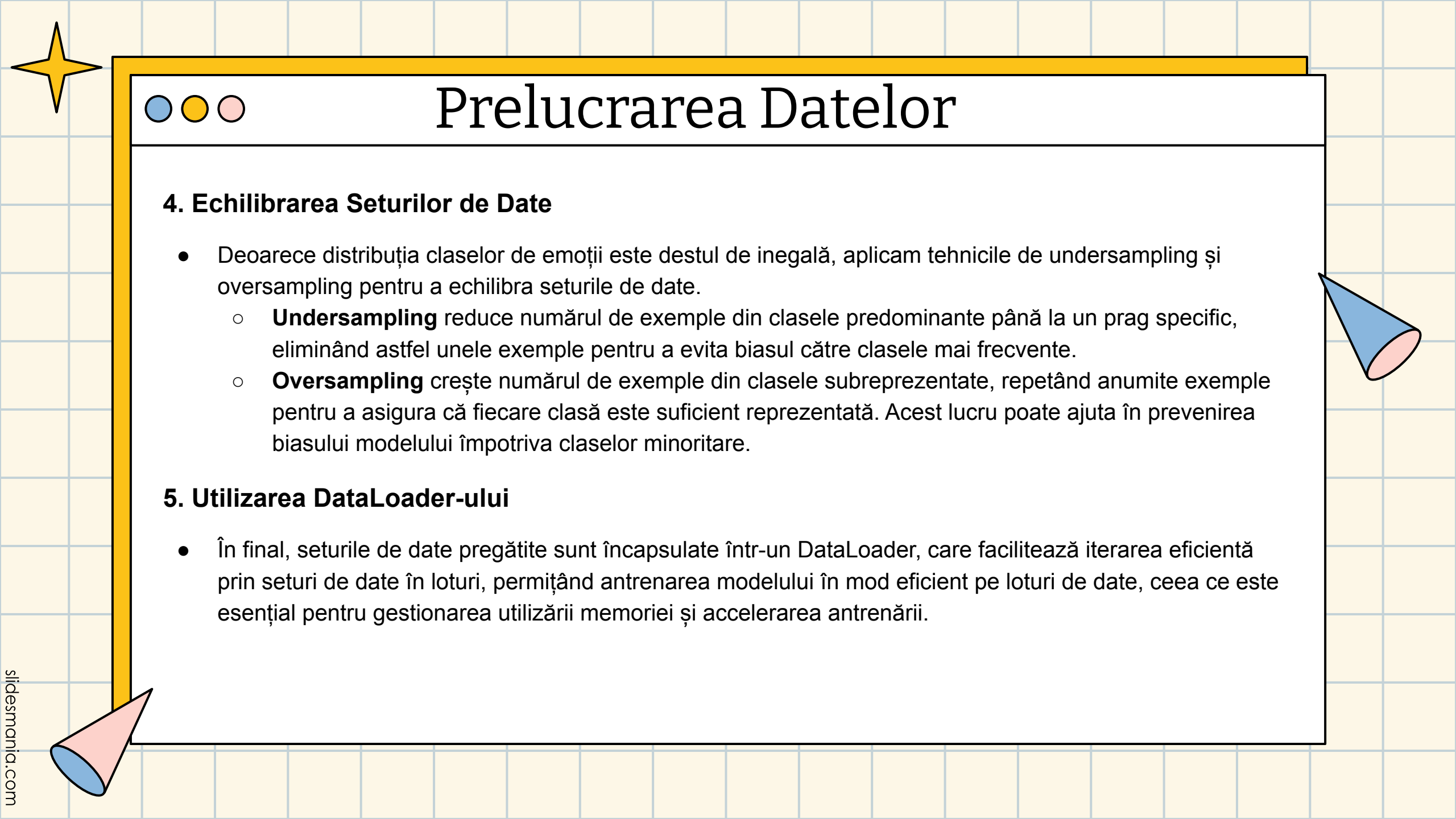
Datele sunt inițial încărcate dintr-un fișier CSV, iar apoi sunt segmentate în seturi de antrenament, validare și testare, bazate pe coloana 'Usage'. Aceasta asigură că modelul poate fi antrenat pe un set de date, validat pe un alt set pentru a ajusta hiperparametrii fără a risca overfitting-ul, și în final testat pe un set complet independent pentru a evalua performanța modelului.

## 2. Preprocesarea Pixelilor

Pixelii fiecărei imagini sunt convertiți din șiruri de caractere în array-uri numpy de valori de pixeli, care sunt apoi normalizați (divizând prin 255.0) pentru a aduce valorile pixelilor în intervalul [0, 1]. Acest pas este crucial pentru a facilita procesul de antrenare, deoarece valorile normalizate sunt mai stabile numeric și ajută la accelerarea convergenței în timpul antrenării.

## 3. Redimensionarea și Transformările Imaginilor

Imaginile sunt convertite la formatul PIL pentru a permite aplicarea de transformări suplimentare. De exemplu, sunt convertite în RGB (chiar dacă sunt monocrome), apoi redimensionate la o dimensiune specifică necesară modelului (de exemplu, 224x224 pentru un model ca ViT) și în cele din urmă transformate în tensori. Normalizarea finală ajustează valorile pixelilor conform unor medii și deviații standard predefinite, care corespund celor folosite în antrenarea prealabilă a modelelor de rețea neurală pe setul de date FER2013.



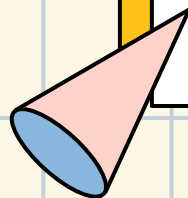
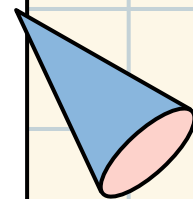
# Prelucrarea Datelor

## 4. Echilibrarea Seturilor de Date

- Deoarece distribuția claselor de emoții este destul de inegală, aplicăm tehnicile de undersampling și oversampling pentru a echilibra seturile de date.
  - **Undersampling** reduce numărul de exemple din clasele predominante până la un prag specific, eliminând astfel unele exemple pentru a evita biasul către clasele mai frecvente.
  - **Oversampling** crește numărul de exemple din clasele subreprezentate, repetând anumite exemple pentru a asigura că fiecare clasă este suficient reprezentată. Acest lucru poate ajuta în prevenirea biasului modelului împotriva claselor minoritare.

## 5. Utilizarea DataLoader-ului

- În final, seturile de date pregătite sunt încapsulate într-un DataLoader, care facilitează iterarea eficientă prin seturi de date în loturi, permițând antrenarea modelului în mod eficient pe loturi de date, ceea ce este esențial pentru gestionarea utilizării memoriei și accelerarea antrenării.





# ResNet101

## Ce este ResNet?

**ResNet (Residual Network)** este o rețea neurală convoluțională care a fost introdusă de Kaiming He și alții în lucrarea lor "Deep Residual Learning for Image Recognition" (2015). ResNet este cunoscută pentru utilizarea "**conexiunilor reziduale**" care permit antrenarea rețelelor foarte profunde fără a suferi de problema dispariției gradientului. Aceste conexiuni reziduale sunt de fapt niște scurtături sau legături care trec peste unul sau mai multe straturi, adăugând output-ul unui strat anterior la un strat ulterior, ceea ce face ca antrenarea să fie mai stabilă și eficientă.

## Descrierea Modelului Personalizat

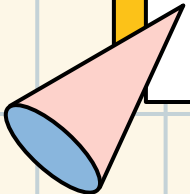
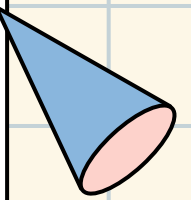
### 1. Adaptarea la Imaginile Monocrom:

Am adaptat primul strat convoluțional al ResNet-101 pentru a accepta imagini monocrome:

- **Ajustarea greutăților:** Primul strat convoluțional (conv1\_conv) în ResNet este configurat inițial pentru a procesa imagini RGB (3 canale). Codul calculează o medie a greutăților pe canale și apoi replică această medie pentru a se potrivi cu structura canalelor RGB, astfel încât să putem folosi ponderile preantrenate de ImageNet chiar și pentru datele monocrome.

### 2. Antrenarea Modelului:

Modelul este antrenat utilizând datele adaptate (X\_train\_rgb) cu label-uri corespunzătoare (y\_train), validând performanța pe un subset de validare.







# ResNet101

## Construirea Modelului Cu Arhitectura ResNet:

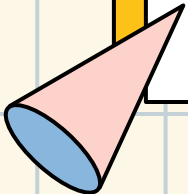
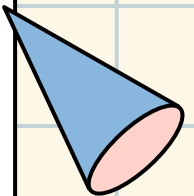
### 2. Constructia Modelului:

De aceasta data construim un model de la zero, utilizând ResNet-101 ca bază:

- **Input Layer:** Definește dimensiunea de intrare a imaginilor (48x48 pixeli, 3 canale).
- **Conv2D Layer:** Adaugă un strat convoluțional suplimentar pentru a adapta modelul la dimensiunea specifică a datelor de intrare. Acest lucru poate ajuta la o mai bună inițializare a rețelei pentru imagini de dimensiuni mici.
- **ResNet101 Layer:** Adaugă ResNet-101 ca un bloc de bază, fără stratul de top, pentru a permite adăugarea de noi straturi specifice sarcinii.
- **GlobalAveragePooling2D:** Reduce dimensiunile spațiale ale hartilor de caracteristici la un vector unic per hartă, ceea ce reduce numărul total de caracteristici și ajută la prevenirea overfitting-ului.
- **Dense Layers:** Include straturi dense (fully connected) care finalizează clasificarea bazată pe caracteristicile extrase de ResNet.

### 2. Antrenarea Modelului:

Modelul este antrenat utilizând datele ( $X_{train}$ ) cu label-uri corespunzătoare ( $y_{train}$ ), validând performanța pe un subset de validare.





# VGG16

## Ce este VGG16?

VGG-16 este o rețea neuronală convoluțională dezvoltată de Karen Simonyan și Andrew Zisserman de la Universitatea Oxford în lucrarea lor "Very Deep Convolutional Networks for Large-Scale Image Recognition". Modelul original VGG-16 a fost proiectat pentru a avea 16 straturi ponderate, și este renumit pentru simplitatea și eficiența sa în clasificarea imaginilor.

## Ce am modificat?

### 1. Batch Normalization:

- **Scop:** Batch Normalization este adăugat pentru a normaliza activările în rețea, ceea ce ajută la accelerarea antrenării prin reducerea numărului de epoci necesare pentru a converge modelul și la stabilizarea procesului de învățare. Normalizarea batch-urilor reduce problema cunoscută sub numele de "internal covariate shift".
- **Implementare:** Fiecare strat convoluțional este urmat de un strat de Batch Normalization, care ajustează activările pentru a avea o medie de zero și o deviație standard de unu.

### 2. Configurația Straturilor Convolutionale:

- **Straturi Conv:** Modelul folosește mai multe straturi convoluționale, fiecare cu filtre de dimensiune 3x3, ceea ce este tipic pentru VGG. Aceste straturi sunt aranjate în blocuri cu număr crescând de filtre de la 64 la 512.
- **Pooling:** Fiecare bloc de straturi convoluționale este urmat de un strat de MaxPooling cu dimensiunea pool-ului de 2x2 și stride de 2, care reduce dimensiunea spațială a hartilor de caracteristici.



# VGG16

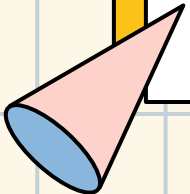
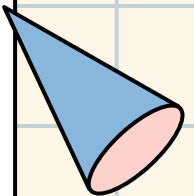
## Ce am modificat?

### 3. Straturi Dense și Dropout:

- **Dense:** La capătul rețelei, două straturi dense cu 4096 de unități fiecare sunt folosite pentru a clasifica caracteristicile extrase.
- **Dropout:** Straturile de Dropout cu o rată de 0.5 sunt utilizate pentru a preveni overfitting-ul prin "dezactivarea" aleatorie a unor unități din straturile dense în timpul antrenamentului.

### 4. Optimizarea și Callback-uri:

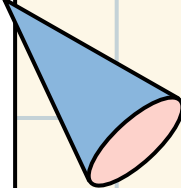
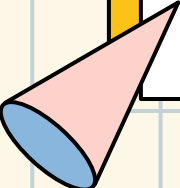
- **ImageDataGenerator:** Folosești ImageDataGenerator pentru a augmenta datele în timpul antrenamentului, introducând modificări ale imaginilor cum ar fi rotații, shift-uri, shear-uri, zoom-uri și flip-uri orizontale. Aceasta ajută la crearea unui model mai robust prin expunerea la diverse variații ale datelor de intrare.
- **ReduceLROnPlateau:** Acest callback reduce rata de învățare când o platou în îmbunătățirea performanței este detectată, ceea ce ajută la finisarea antrenării în zonele minime ale spațiului de căutare.
- **ModelCheckpoint:** Salvează cea mai bună versiune a modelului bazată pe acuratețea de validare, asigurându-se că ai acces la modelul cu performanța cea mai bună la sfârșitul antrenării.
- **EarlyStopping:** Oprește prematură antrenamentul dacă performanța nu se îmbunătățește după un număr definit de epoci, pentru a evita risipa de resurse computaționale și overfitting-ul.





# Custom CNN

## Straturile Convolutionale și MaxPooling

1. **Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=input\_shape):**
    - Acesta este primul strat convoluțional.
    - **32** indică numărul de filtre utilizate, adică modelul va învăța 32 de hărți diferite de caracteristici la acest nivel.
    - **kernel\_size=(3, 3)** definește dimensiunea fiecărui filtru ca fiind 3x3.
    - **activation='relu'** specifică utilizarea funcției de activare ReLU (Rectified Linear Unit), care adaugă non-linearitate procesului de învățare, ajutând la captarea modelelor complexe în date.
    - **input\_shape=input\_shape** indică forma datelor de intrare pe care le așteaptă stratul, în acest caz, imagini de 48x48 pixeli cu un singur canal (tonuri de gri).
  2. **MaxPooling2D(pool\_size=(2, 2)):**
    - Aceste straturi urmează fiecare strat convoluțional și servesc la reducerea dimensiunilor spațiale ale hărților de caracteristici.
    - **pool\_size=(2, 2)** indică dimensiunea ferestrei de pooling; modelul ia maximum din fiecare fereastră de 2x2 pixeli, reducând în jumătate lățimea și înălțimea hărților de caracteristici.
  3. **Conv2D(64, (3, 3), activation='relu')** și **Conv2D(128, (3, 3), activation='relu'):**
    - Aceste straturi urmează aceeași logică ca primul strat convoluțional, dar cu un număr crescând de filtre (64 și apoi 128). Acest lucru permite modelului să detecteze caracteristici din ce în ce mai complexe pe măsură ce datele progresează prin rețea.
- 
- 



# Custom CNN

## Stratul Flatten și Straturile Dense

4. **Flatten():**
  - Acest strat transformă hărțile de caracteristici 2D rezultate din ultimul strat de pooling într-un vector 1D. Aceasta este necesar pentru a putea folosi straturi dense, care prelucrează datele sub formă de vectori.
5. **Dense(1024, activation='relu'):**
  - Un strat dens (fully-connected) cu 1024 de unități, care poate învăța o reprezentare complexă a datelor extrase de straturile convoluționale și de pooling.
  - Funcția de activare ReLU este folosită din nou pentru a introduce non-linearități în procesul de învățare.
6. **Dropout(0.5):**
  - Acest strat "dezactivează" aleatoriu jumătate din unitățile din stratul anterior pe parcursul antrenamentului, ceea ce ajută la prevenirea overfitting-ului.
7. **Dense(num\_classes, activation='softmax'):**
  - Ultimul strat este un alt strat dens, cu un număr de unități egal cu numărul de clase de emoții (7).
  - Utilizează funcția de activare softmax pentru a produce o distribuție de probabilitate peste clasele de ieșire, facilitând clasificarea.



# Custom RNN

## De ce RNN?

În general, un model RNN este destinat să proceseze secvențe de date, ceea ce este diferit de modul tradițional în care CNN-urile tratează imagini. Prin tratarea fiecărui rând de pixeli dintr-o imagine ca un "pas de timp" într-o secvență, modelul poate învăța dependențe temporale între rândurile de pixeli ale unei imagini, ceea ce poate fi util în recunoașterea unor modele de forme și texturi care sunt specifice diferitelor emoții.

## Ce am utilizat?

### 1. Straturi LSTM Bidirecționale:

- **Bidirectional(LSTM(64, return\_sequences=True))**: Acesta este primul strat LSTM, care procesează secvența de la început la sfârșit și de la sfârșit la început simultan. Acesta returnează secvențe complete (nu doar ultimul output), ceea ce înseamnă că fiecare pas de timp va avea un output corespunzător. Numărul 64 reprezintă dimensiunea spațiului de ieșire al stratului, adică numărul de unități în LSTM.
- **Bidirectional(LSTM(64))**: Al doilea strat LSTM bidirecțional nu are `return_sequences=True`, deci va returna doar ultimul output pentru fiecare secvență, rezumând astfel întreaga secvență la un singur vector de caracteristici.

### 2. Straturi Dense:

- **Dense(64, activation='relu')**: Aceasta este un strat dens cu 64 de unități, care primește vectorul de caracteristici de la LSTM și continuă procesul de învățare, adăugând non-linearitate prin activarea ReLU.
- **Dense(num\_classes, activation='softmax')**: Ultimul strat dens are un număr de unități egal cu numărul de clase (7 emoții) și folosește activarea softmax pentru a transforma scorurile logit în probabilități pentru fiecare clasă.



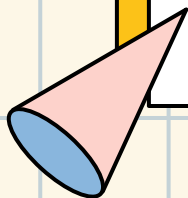
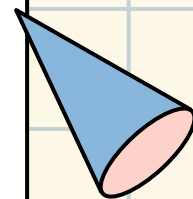
# Custom RNN

## Compilarea Modelului

Modelul este compilat cu optimizatorul 'adam' și folosește pierderea 'categorical\_crossentropy', care este adecvată pentru problemele de clasificare multi-clasă, având ca metrică 'accuracy'.

## Antrenarea Modelului

- Modelul este antrenat folosind datele de antrenament cu etichetele corespunzătoare. În timpul antrenamentului, se utilizează și un set de validare pentru a monitoriza pierderea și acuratețea pe date nevăzute anterior.
- **Callbacks:**
  - **EarlyStopping:** Urmărește pierderea pe setul de validare și oprește antrenamentul dacă aceasta nu se îmbunătățește pentru un număr de 5 epoci consecutive. În plus, acest callback restaurează greutatea modelului la cele mai bune valori observate, ajutând la prevenirea overfitting-ului.



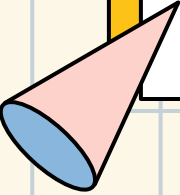
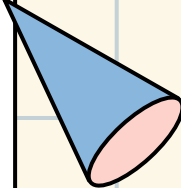


# Custom ViT

## Ce este ViT?

Vision Transformer (ViT) este o arhitectură în domeniul procesării imaginilor, care aplică principiile transformerilor, originar dezvoltate pentru procesarea limbajului natural, la viziunea computerizată. A fost introdus de cercetătorii de la Google AI în lucrarea lor din 2020 intitulată "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale".

## Preprocesarea Datelor

1. **Transformări:** Se folosesc mai multe transformări pentru a adapta imaginile la cerințele modelului ViT:
    - **Convertirea la uint8:** Asigură că datele sunt în formatul corect pentru procesare ulterioară.
    - **Conversie la PIL Image:** Transformă array-urile numpy în obiecte de tip PIL Image, care sunt mai ușor de manipulat pentru transformări ulterioare.
    - **Convertirea la RGB:** Chiar dacă imaginile sunt monocrome, modelul ViT așteaptă intrări RGB, deci imaginile sunt convertite în acest format.
    - **Redimensionare:** Imaginile sunt scalate la dimensiunea de 224x224 pixeli, care este dimensiunea standard pentru ViT.
    - **Conversie în Tensor:** Imaginile sunt convertite în tensori PyTorch.
    - **Normalizare:** Datele sunt normalizate folosind medii și deviații standard specifice, ceea ce este standard pentru modelele preantrenate.
- 
- 





# Custom ViT

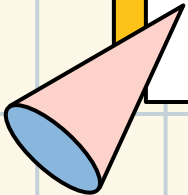
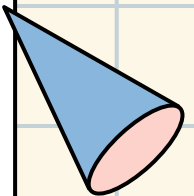
## Încărcarea Datelor

2. **DataLoaders:** Se folosesc pentru a itera prin seturile de date în loturi, asigurând amestecarea datelor și încărcarea eficientă în timpul antrenamentului și evaluării.

## Configurarea Modelului Vision Transformer

3. **ViT Configurație și Model:**

- **Config:** Se încarcă configurația de la un model ViT preantrenat, ajustându-l pentru a avea un număr de etichete corespunzător numărului de clase.
- **Model:** Se încarcă un model ViT preantrenat, configurat pentru a clasifica imaginile în 7 clase diferite de emoții.





# Custom ViT

## Antrenarea Modelului

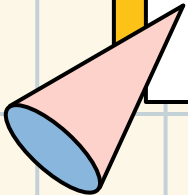
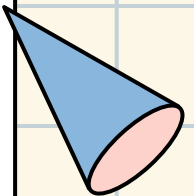
4. **TrainingArguments:** Configurarea argumentelor pentru antrenament, incluzând directorul de output, numărul de epoci, dimensiunea loturilor, logarea, strategia de evaluare și salvare, și altele.

5. **Trainer:**

**Trainer Setup:** O instanță de Trainer care gestionează ciclul de antrenament și evaluare, folosind modelul și argumentele specificate.

**Compute Metrics:** O funcție care calculează metricile de performanță pentru evaluare, utilizând acuratețea și scorul F1

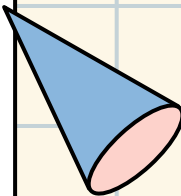
6. **Procesul de Antrenament:** Modelul este antrenat folosind datele pregătite, cu callback-uri pentru oprirea antrenamentului la timp pentru a preveni overfitting-ul și pentru a salva cel mai bun model în funcție de acuratețe.







# Rezultate

Model	Acuratete Antrenament	Acuratete Validare	Acuratete Test
VGG16	0.7229	<b>0.6723</b>	<b>0.6801</b>
ResNet101	<b>0.9656</b>	0.5618	0.5497
Custom CNN	0.7122	0.5795	0.5834
Custom RNN	0.7369	0.4461	0.4521
Custom ViT	0.7005	0.569	0.5621





# Link-uri



**HuggingFace:** <https://huggingface.co/AndreiUrsu>

**Cod Complet Proiect:** <https://colab.research.google.com/drive/13EvTvP-M5LtyTtO-4KdsMhJpXiJfoCiQ?usp=sharing>

**Demo:** [https://colab.research.google.com/drive/15fn8sx2V72FNEIx4ijXk\\_qkJrCCIV8Ca?usp=sharing](https://colab.research.google.com/drive/15fn8sx2V72FNEIx4ijXk_qkJrCCIV8Ca?usp=sharing)

