الغاية من الجلسة:

- التعرف على CNN MODEL
- SOME TYPES OF OPTIMIZERS

Face Recognition with CNN

```
import tensorflow as tf
import numpy as np
import cv2
import os
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
```

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("vasukipatel/face-recognition-dataset")

print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/vasukipatel/face-recognition-dataset?dataset_version_number=1...
100%|██████████| 726M/726M [00:09<00:00, 83.4MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/vasukipatel/face-recognition-dataset/versions/1
```

```
train_dir="/root/.cache/kagglehub/datasets/vasukipatel/face-recognition-dataset/versions/1/Original Images/Original Images/"
```

```
generator = ImageDataGenerator()
train_ds = generator.flow_from_directory(train_dir,target_size=(224, 224),batch_size=32)
classes = list(train_ds.class_indices.keys())
```

```
Found 2562 images belonging to 31 classes.
```

```
fig = plt.figure(figsize=(10,5))
src_path="/root/.cache/kagglehub/datasets/vasukipatel/face-recognition-dataset/versions/1/Original Images/Original Images/Akshay Kumar/Aksha

img = plt.imread(src_path)
plt.imshow(img)
```

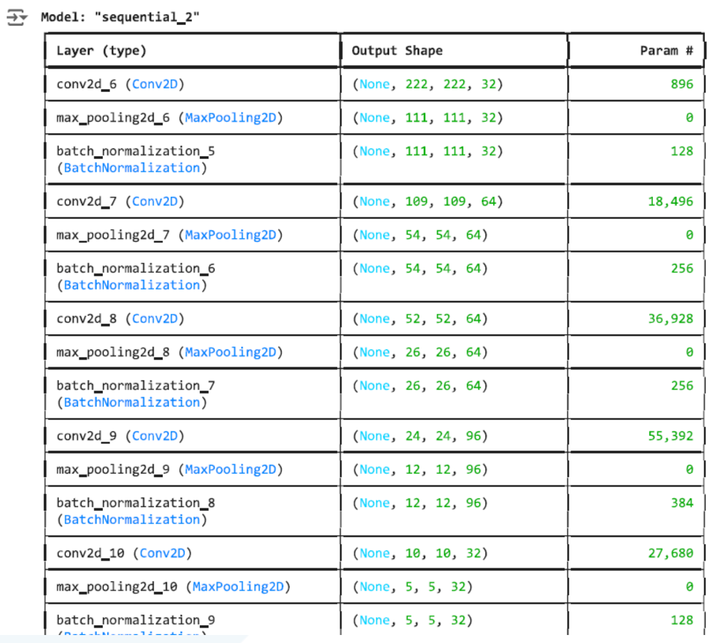
```
<matplotlib.image.AxesImage at 0x799cf0f0e3e0>
```

```
# import numpy as np
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
import matplotlib.pyplot as plt
# from keras.layers.normalization import BatchNormalization
```

```
from tensorflow.keras.layers import BatchNormalization


model = Sequential()
model.add(Conv2D(32, kernel_size = (3, 3), activation='relu', input_shape=(224,224,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(96, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
#model.add(Dropout(0.3))
model.add(Dense(len(classes),activation='softmax'))


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



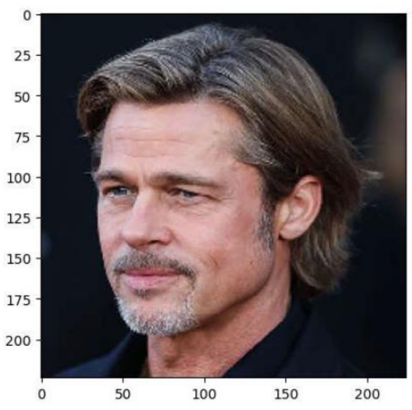
```
Model: "sequential_2"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape                 ┃        Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━┩
│ conv2d_6 (Conv2D)               │ (None, 222, 222, 32)         │            896 │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ max_pooling2d_6 (MaxPooling2D)  │ (None, 111, 111, 32)         │              0 │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ batch_normalization_5           │ (None, 111, 111, 32)         │            128 │
│ (BatchNormalization)            │                              │                │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ conv2d_7 (Conv2D)               │ (None, 109, 109, 64)         │         18,496 │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ max_pooling2d_7 (MaxPooling2D)  │ (None, 54, 54, 64)           │              0 │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ batch_normalization_6           │ (None, 54, 54, 64)           │            256 │
│ (BatchNormalization)            │                              │                │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ conv2d_8 (Conv2D)               │ (None, 52, 52, 64)           │         36,928 │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ max_pooling2d_8 (MaxPooling2D)  │ (None, 26, 26, 64)           │              0 │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ batch_normalization_7           │ (None, 26, 26, 64)           │            256 │
│ (BatchNormalization)            │                              │                │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ conv2d_9 (Conv2D)               │ (None, 24, 24, 96)           │         55,392 │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ max_pooling2d_9 (MaxPooling2D)  │ (None, 12, 12, 96)           │              0 │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ batch_normalization_8           │ (None, 12, 12, 96)           │            384 │
│ (BatchNormalization)            │                              │                │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ conv2d_10 (Conv2D)              │ (None, 10, 10, 32)           │         27,680 │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ max_pooling2d_10 (MaxPooling2D) │ (None, 5, 5, 32)             │              0 │
├─────────────────────────────────┼──────────────────────────────┼────────────────┤
│ batch_normalization_9           │ (None, 5, 5, 32)             │            128 │
│ max_pooling2d_7 (MaxPooling2D)       │ (None, 54, 54, 64)          │              0 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ batch_normalization_6                │ (None, 54, 54, 64)          │            256 │
│ (BatchNormalization)                 │                             │                │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ conv2d_8 (Conv2D)                    │ (None, 52, 52, 64)          │         36,928 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ max_pooling2d_8 (MaxPooling2D)       │ (None, 26, 26, 64)          │              0 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ batch_normalization_7                │ (None, 26, 26, 64)          │            256 │
│ (BatchNormalization)                 │                             │                │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ conv2d_9 (Conv2D)                    │ (None, 24, 24, 96)          │         55,392 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ max_pooling2d_9 (MaxPooling2D)       │ (None, 12, 12, 96)          │              0 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ batch_normalization_8                │ (None, 12, 12, 96)          │            384 │
│ (BatchNormalization)                 │                             │                │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ conv2d_10 (Conv2D)                   │ (None, 10, 10, 32)          │         27,680 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ max_pooling2d_10 (MaxPooling2D)      │ (None, 5, 5, 32)            │              0 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ batch_normalization_9                │ (None, 5, 5, 32)            │            128 │
│ (BatchNormalization)                 │                             │                │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ dropout_1 (Dropout)                  │ (None, 5, 5, 32)            │              0 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ flatten_1 (Flatten)                  │ (None, 800)                 │              0 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ dense_2 (Dense)                      │ (None, 128)                 │        102,528 │
├──────────────────────────────────────┼─────────────────────────────┼────────────────┤
│ dense_3 (Dense)                      │ (None, 31)                  │          3,999 │
└──────────────────────────────────────┴─────────────────────────────┴────────────────┘
 Total params: 247,071 (965.12 KB)
 Trainable params: 246,495 (962.87 KB)
 Non-trainable params: 576 (2.25 KB)
```

```python
history = model.fit(train_ds,epochs= 3, batch_size=32)
```

```
Epoch 1/3
81/81 ━━━━━━━━━━━━━━━━━━━━ 313s 4s/step - accuracy: 0.8192 - loss: 0.6978
Epoch 2/3
81/81 ━━━━━━━━━━━━━━━━━━━━ 307s 4s/step - accuracy: 0.8244 - loss: 0.6460
Epoch 3/3
81/81 ━━━━━━━━━━━━━━━━━━━━ 329s 4s/step - accuracy: 0.8223 - loss: 0.5861
```

```python
def predict_image(image_path):
    img = image.load_img(image_path, target_size=(224,224,3))
    plt.imshow(img)
    plt.show()
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    images = np.vstack([x])
    pred = model.predict(images, batch_size=32)
    print("Actual: "+(image_path.split("/")[-1]).split("_")[0])
    print("Predicted: "+classes[np.argmax(pred)])
```

```python
predict_image("/root/.cache/kagglehub/datasets/vasukipatel/face-recognition-dataset/versions/1/Original Images/Original Images/Brad Pitt/Bra
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
Actual: Brad Pitt
Predicted: Brad Pitt
```

```python
predict_image("/root/.cache/kagglehub/datasets/vasukipatel/face-recognition-dataset/versions/1/Original Images/Original Images/Henry Cavill/
```

Types of Optimizers

Stochastic Gradient descent

tochastic Gradient Descent (SGD) is an iterative optimization algorithm commonly used in machine learning and deep learning. It is a variant of gradient descent that performs updates to the model parameters (weights) based on the gradient of the loss function computed on a randomly selected subset of the training data, rather than on the full dataset.

The basic idea of SGD is to sample a small random subset of the training data, called a mini-batch, and compute the gradient of the loss function with respect to the model parameters using only that subset. This gradient is then used to update the parameters. The process is repeated with a new random mini-batch until the algorithm converges or reaches a predefined stopping criterion.

SGD has several advantages over standard gradient descent, such as faster convergence and lower memory requirements, especially for large datasets. It is also more robust to noisy and non-stationary data, and can escape from local minima. However, it may require more iterations to

converge than gradient descent, and the learning rate needs to be carefully tuned to ensure convergence.

Stochastic Gradient descent with gradient clipping

Stochastic Gradient Descent with gradient clipping (SGD with GC) is a variant of the standard SGD algorithm that includes an additional step to prevent the gradients from becoming too large during training, which can cause instability and slow convergence.

Gradient clipping involves scaling down the gradients if their norm exceeds a predefined threshold. This helps to prevent the "exploding gradient" problem, which can occur when the gradients become too large and cause the weights to update too much in a single step.

In SGD with GC, the algorithm computes the gradients on a randomly selected mini-batch of training examples, as in standard SGD. However, before applying the gradients to update the model parameters, the gradients are clipped if their norm exceeds a specified threshold. This threshold is typically set to a small value, such as 1.0 or 5.0.

The gradient clipping step can be applied either before or after any regularization techniques, such as L2 regularization. It is also common to use adaptive learning rate algorithms, such as Adam, in conjunction with SGD with GC to further improve convergence.

SGD with GC is particularly useful when training deep neural networks, where the gradients can easily become unstable and cause convergence problems. By limiting the size of the gradients, the algorithm can converge faster and with greater stability, leading to improved performance on the test set.

Momentum

Momentum is an optimization technique used in machine learning and deep learning to accelerate the training of neural networks. It is based on the idea of adding a fraction of the previous update to the current update of the weights during the optimization process.

In momentum optimization, the gradient of the cost function is computed with respect to each weight in the neural network. Instead of updating the weights directly based on the gradient, momentum optimization introduces a new variable, called the momentum term, which is used to update the weights. The momentum term is a moving average of the gradients, and it accumulates the past gradients to help guide the search direction.

The momentum term can be interpreted as the velocity of the optimizer. The optimizer accumulates momentum as it moves downhill and helps to dampen oscillations in the optimization process. This can help the optimizer to converge faster and to reach a better local minimum.

Momentum optimization is particularly useful in situations where the optimization landscape is noisy or where the gradients change rapidly. It can also help to smooth out the optimization process and prevent the optimizer from getting stuck in local minima.

Overall, momentum is a powerful optimization technique that can help accelerate the training of deep neural networks and improve their performance.

Adagrad

Adagrad (Adaptive Gradient) is an optimization algorithm used in machine learning and deep learning to optimize the training of neural networks.

The Adagrad algorithm adjusts the learning rate of each parameter of the neural network adaptively during the training process. Specifically, it scales the learning rate of each parameter based on the historical gradients computed for that parameter. In other words, parameters that have large gradients are given a smaller learning rate, while those with small gradients are given a larger learning rate. This helps prevent the learning rate from decreasing too quickly for frequently occurring parameters and allows for faster convergence of the training process.

The Adagrad algorithm is particularly useful for dealing with sparse data, where some of the input features have low frequency or are missing. In these cases, Adagrad is able to adaptively adjust the learning rate of each parameter, which allows for better handling of the sparse data.

Overall, Adagrad is a powerful optimization algorithm that can help accelerate the training of deep neural networks and improve their performance

Adadelta

Adadelta is an optimization algorithm used in machine learning and deep learning to optimize the training of neural networks. It is a variant of the Adagrad algorithm and addresses some of its limitations.

The Adadelta algorithm adapts the learning rate of each parameter in a similar way to Adagrad, but instead of storing all the past gradients, it only stores a moving average of the squared gradients. This helps to reduce the memory requirements of the algorithm.

Additionally, Adadelta uses a technique called "delta updates" to adjust the learning rate. Instead of using a fixed learning rate, Adadelta uses the ratio of the root mean squared (RMS) of the past gradients and the RMS of the past updates to scale the learning rate. This helps to further prevent the learning rate from decreasing too quickly for frequently occurring parameters.

RMSProp

RMSProp (Root Mean Square Propagation) is an optimization algorithm used in machine learning and deep learning to optimize the training of neural networks.

Like Adagrad and Adadelta, RMSProp adapts the learning rate of each parameter during the training process. However, instead of accumulating all the past gradients like Adagrad, RMSProp computes a moving average of the squared gradients. This allows the algorithm to adjust the learning rate more smoothly, and it prevents the learning rate from decreasing too quickly.

The RMSProp algorithm also uses a decay factor to control the influence of past gradients on the learning rate. This decay factor allows the algorithm to give more weight to recent gradients and less weight to older gradients.

One of the main advantages of RMSProp over Adagrad is that it can handle non-stationary objectives, where the underlying function that the neural network is trying to approximate changes over time. In these cases, Adagrad may converge too quickly, but RMSProp can adapt the learning rate to the changing objective function.

Overall, RMSProp is a powerful optimization algorithm that can help accelerate the training of deep neural networks and improve their performance, particularly in situations where the objective function is non-stationary.

Adam

Adam (Adaptive Moment Estimation) is an optimization algorithm used in machine learning and deep learning to optimize the training of neural networks.

Adam combines the concepts of both momentum and RMSProp. It maintains a moving average of the gradient's first and second moments, which are the mean and variance of the gradients, respectively. The moving average of the first moment, which is similar to the momentum