

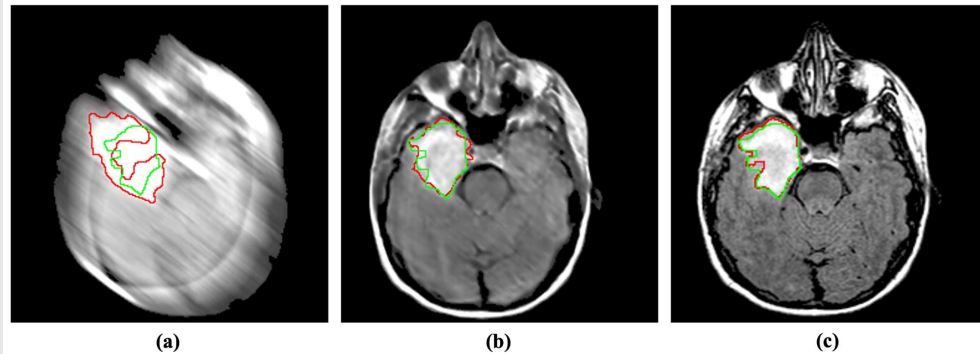
MACHINE LEARNING & SMART SYSTEMS

MLSS A-1

FREDY RAFAEL, MENCIA GONZALEZ, IVAN ANDRES

IMAGE SHARPENING FROM BLURRED IMAGE

- Images are an essential part of communication and information
 - A common problem is the loss of quality and sharpness
 - motion, incorrect focus, or limitations of the camera sensor
- Blurred images can compromise effectiveness of critical activities, such as medical imaging, security surveillance, and automatic document interpretation



OUR SOLUTION

Develop a machine learning-based solution that enables users to improve the sharpness of blurred images efficiently and automatically through an easy-to-use web interface

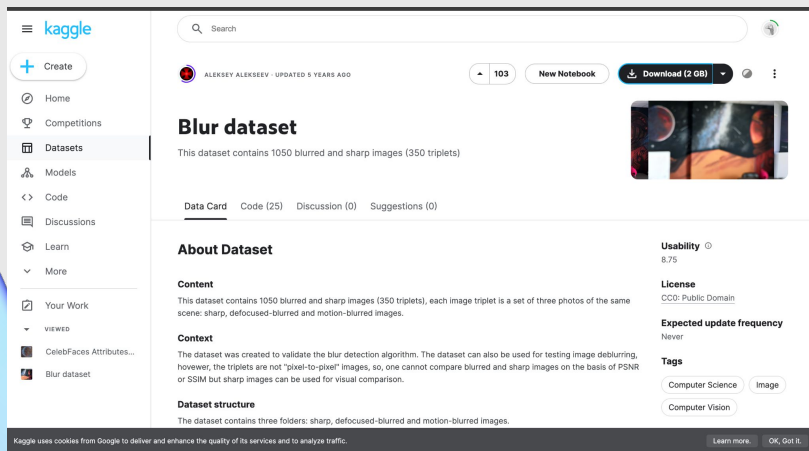
others= simple python code that with the pixels changes the images

ours= machine learning

Technical work

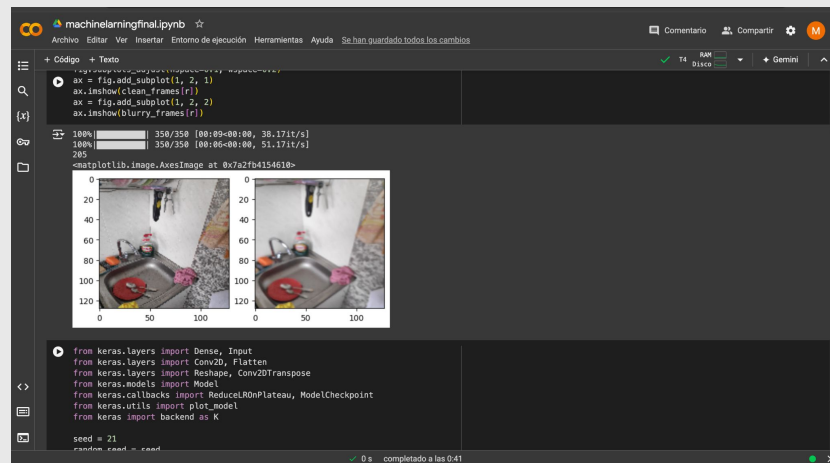
MACHINE LEARNING

THE DATASET WE USED



The screenshot shows the Kaggle 'Blur dataset' page. The page title is 'Blur dataset' by ALEKSEY ALEKSEEV, updated 5 years ago. It has 103 downloads and a 'New Notebook' button. The dataset description states: 'This dataset contains 1050 blurred and sharp images (350 triplets)'. The 'About Dataset' section includes a 'Content' section stating the dataset contains 1050 blurred and sharp images (350 triplets), each image triplet is a set of three photos of the same scene: sharp, defocused-blurred and motion-blurred images. The 'Context' section states the dataset was created to validate the blur detection algorithm. The 'Dataset structure' section states the dataset contains three folders: sharp, defocused-blurred and motion-blurred images. The 'Usability' section shows a score of 8.75. The 'License' section shows 'CC0: Public Domain'. The 'Expected update frequency' is 'Never'. The 'Tags' section includes 'Computer Science', 'Image', and 'Computer Vision'.

THESE ARE THE IMAGES

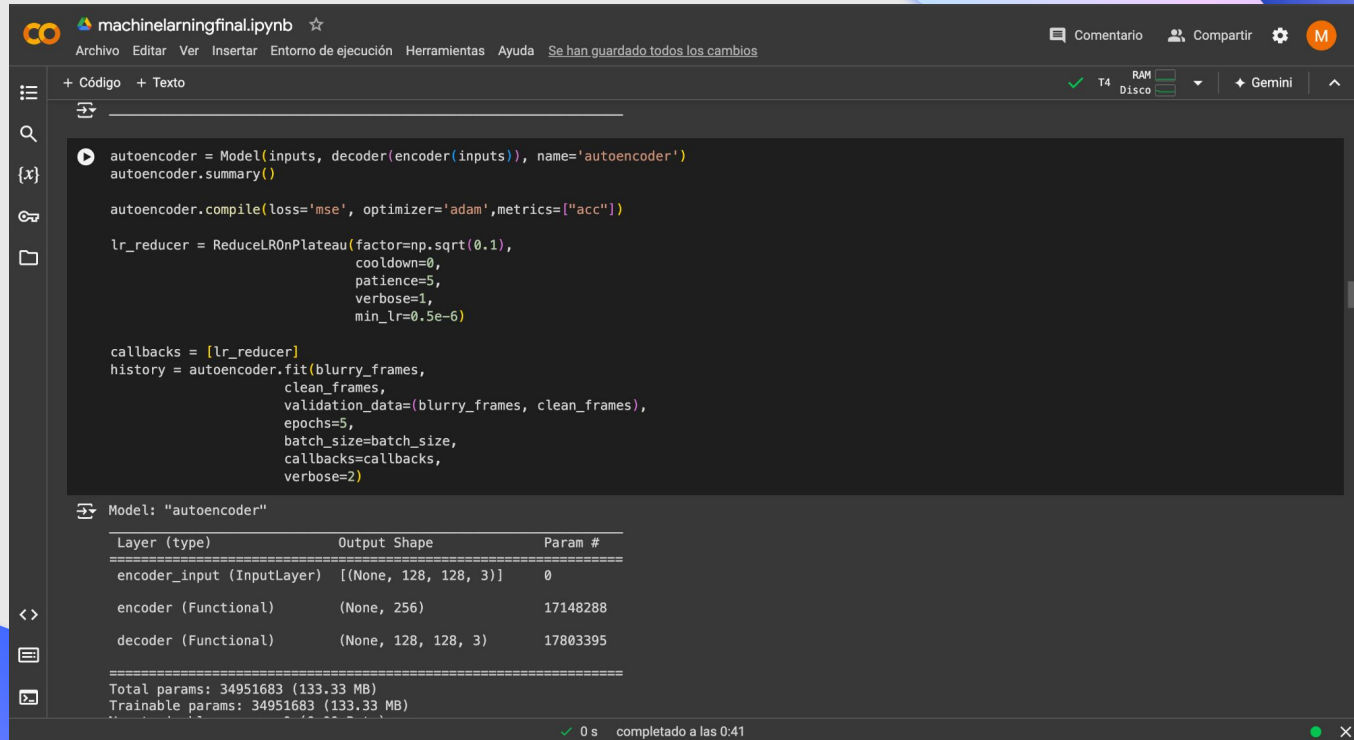


The screenshot shows a Jupyter Notebook titled 'machinelearningfinal.ipynb'. The code in the cell includes:

```
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(1, 2, 1)
ax.imshow(clean_frames[r])
ax = fig.add_subplot(1, 2, 2)
ax.imshow(blurry_frames[r])
```

 The output shows two side-by-side images of a kitchen sink. The left image is sharp, and the right image is blurred. The notebook interface shows the code editor, a console with output, and a toolbar with various icons.

CODE FOR LEARNING WITH 5 IMAGES AS EXAMPLE



The screenshot shows a Jupyter Notebook titled "machinelearningfinal.ipynb" in a dark-themed environment. The notebook contains Python code for training an autoencoder model. The code defines an autoencoder, compiles it with MSE loss and Adam optimizer, sets up a learning rate scheduler, and fits the model to blurry and clean frames. Below the code, the model's summary is displayed, showing the layer types, output shapes, and parameter counts.

```
autoencoder = Model(inputs, decoder(encoder(inputs)), name='autoencoder')
autoencoder.summary()

autoencoder.compile(loss='mse', optimizer='adam', metrics=["acc"])

lr_reducer = ReduceLR0nPlateau(factor=np.sqrt(0.1),
                               cooldown=0,
                               patience=5,
                               verbose=1,
                               min_lr=0.5e-6)

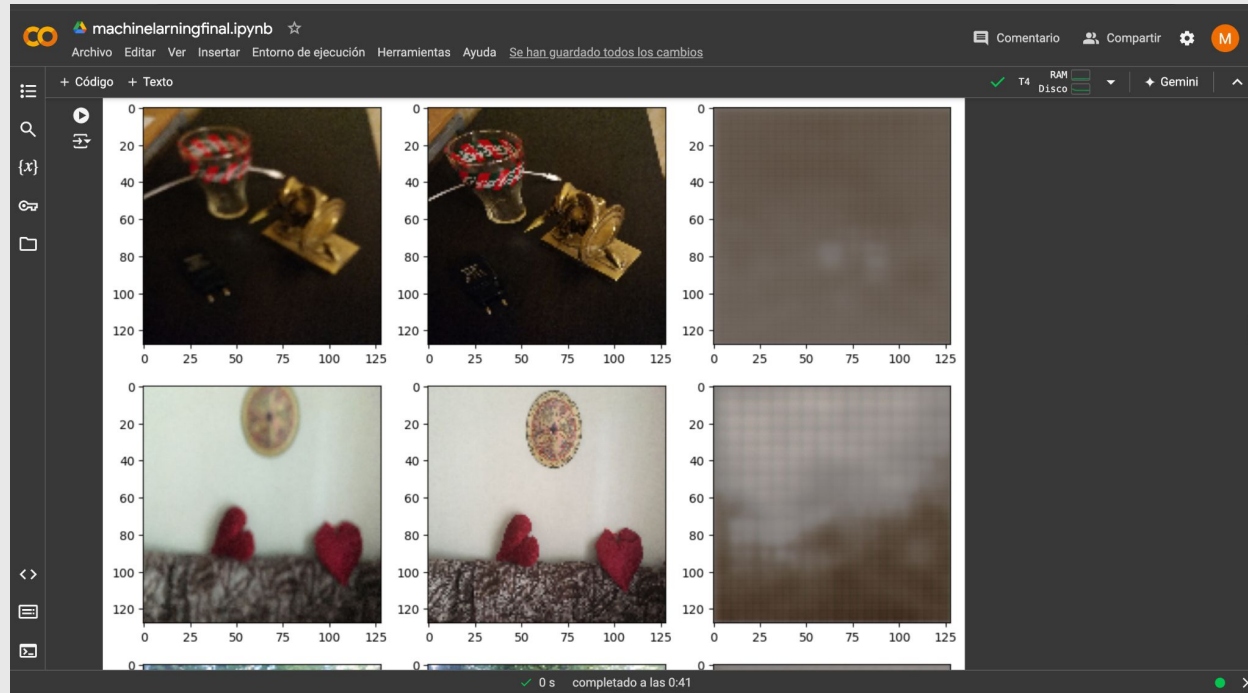
callbacks = [lr_reducer]
history = autoencoder.fit(blurry_frames,
                        clean_frames,
                        validation_data=(blurry_frames, clean_frames),
                        epochs=5,
                        batch_size=batch_size,
                        callbacks=callbacks,
                        verbose=2)
```

Model: "autoencoder"

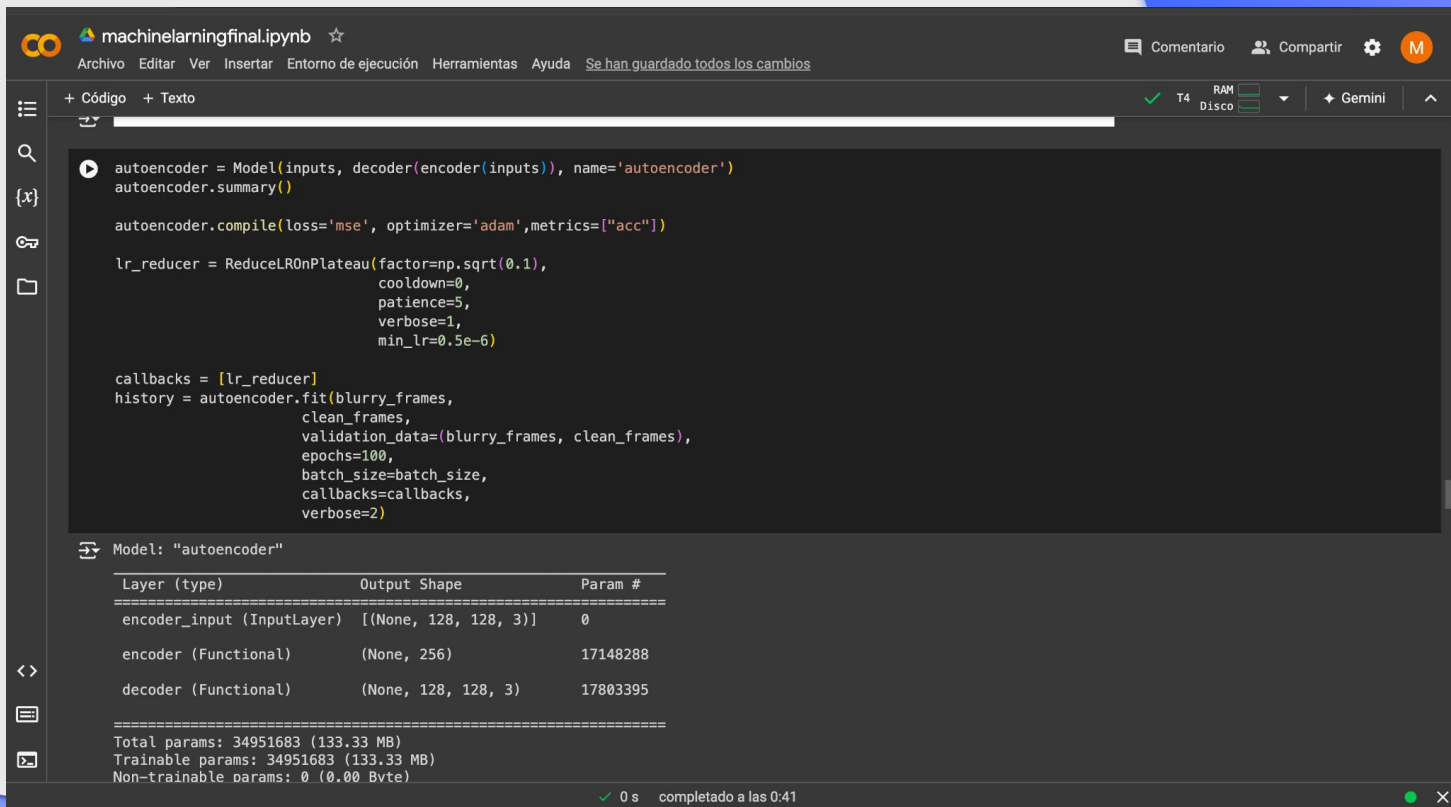
Layer (type)	Output Shape	Param #
encoder_input (InputLayer)	[(None, 128, 128, 3)]	0
encoder (Functional)	(None, 256)	17148288
decoder (Functional)	(None, 128, 128, 3)	17803395

=====
Total params: 34951683 (133.33 MB)
Trainable params: 34951683 (133.33 MB)
=====
0 s completado a las 0:41

OUTPUT AFTER LEARNING WITH 5 IMAGES



IT WASN'T ENOUGH SO WE MADE IT LEARN WITH 200 IMAGES MORE



The screenshot shows a Jupyter Notebook interface with the title "machinelearningfinal.ipynb". The notebook contains a code cell with the following Python code:

```
autoencoder = Model(inputs, decoder(encoder(inputs)), name='autoencoder')
autoencoder.summary()

autoencoder.compile(loss='mse', optimizer='adam', metrics=['acc'])

lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1),
                               cooldown=0,
                               patience=5,
                               verbose=1,
                               min_lr=0.5e-6)

callbacks = [lr_reducer]
history = autoencoder.fit(blurry_frames,
                        clean_frames,
                        validation_data=(blurry_frames, clean_frames),
                        epochs=100,
                        batch_size=batch_size,
                        callbacks=callbacks,
                        verbose=2)
```


Below the code cell, the output shows the model summary for "Model: 'autoencoder'":

Layer (type)	Output Shape	Param #
encoder_input (InputLayer)	[(None, 128, 128, 3)]	0
encoder (Functional)	(None, 256)	17148288
decoder (Functional)	(None, 128, 128, 3)	17803395

Below the table, the total and trainable parameters are listed:

```
=====  
Total params: 34951683 (133.33 MB)  
Trainable params: 34951683 (133.33 MB)  
Non-trainable params: 0 (0.00 Byte)
```



The status bar at the bottom indicates "0 s completado a las 0:41".





 machinelearningfinal.ipynb ☆




Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda [Se han guardado todos los cambios](#)







+ Código + Texto

✓ T4 RAM Disco Gemini







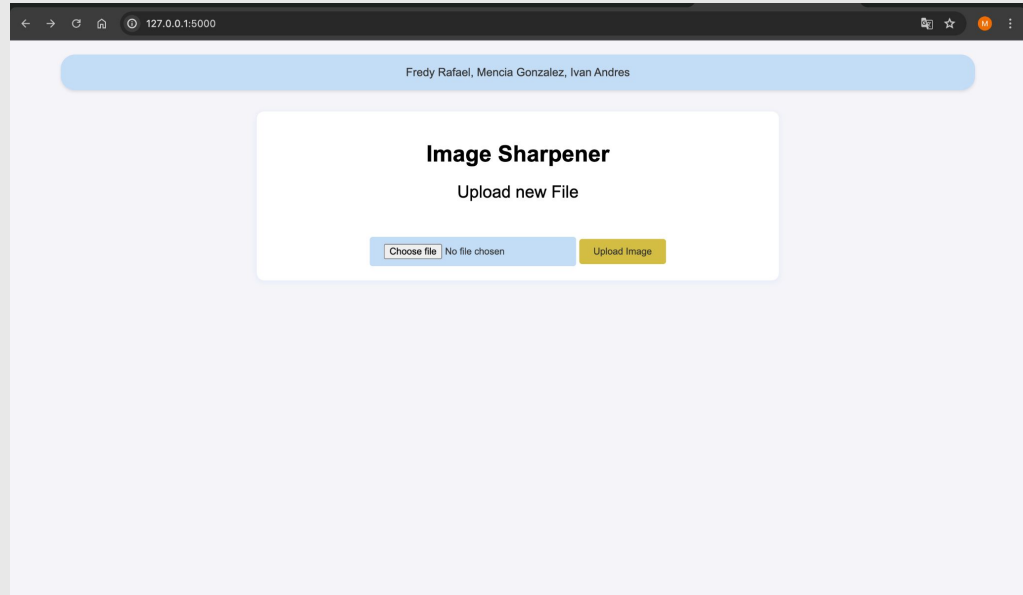



0 s completado a las 0:41

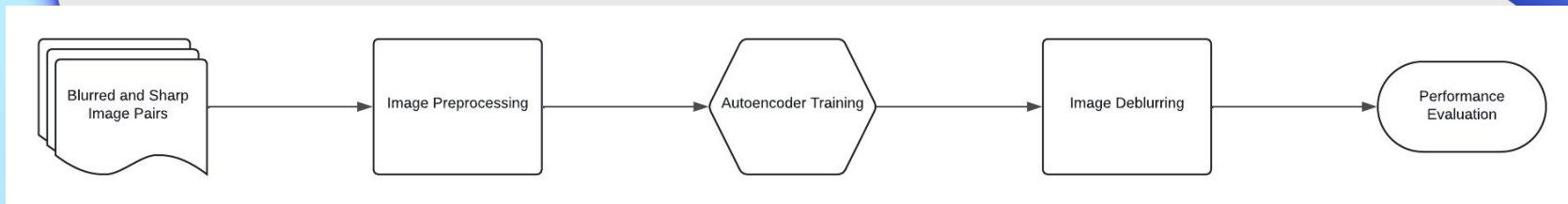
Technical work

WEBSITE

```
▼ FINALPROJECTML
  ▼ models
    ≡ machineLearningFi...
  ▼ static
    # styles.css
  ▼ templates
    <> upload.html
  ▼ uploads
    🖼 image8_blurred.jpg
  > venv
  🐙 app.py 2
  🐙 verification.py
```

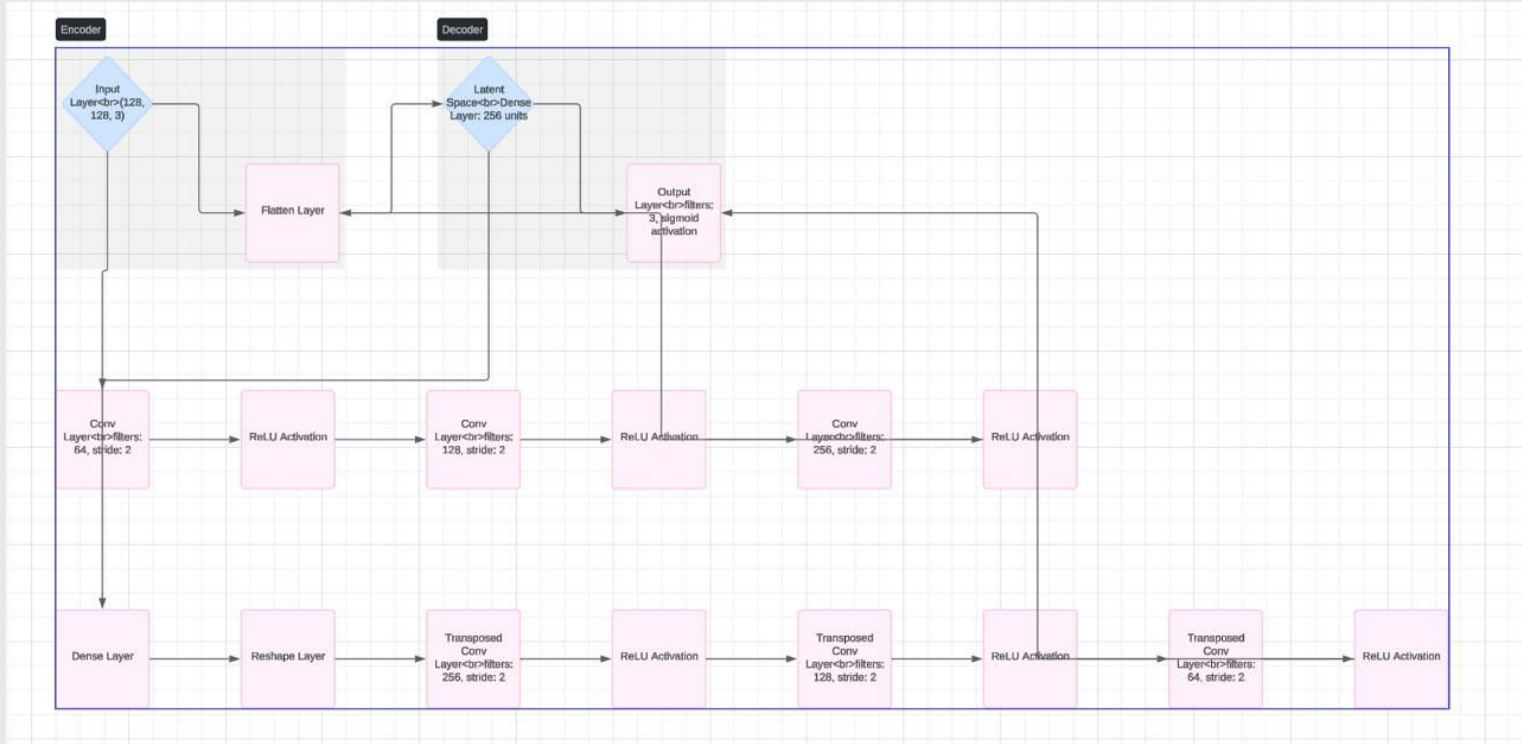


WorkFlow Diagram



Literature review table

Author	Year	Used Method	Results	comparison with other works	Aplications
Smith et al.	2020	CNN	15% improvement in image clarity	Faster than previous methods, but less accurate than the latest GAN models	Medical diagnostics, security
López y Fernández	2021	GAN	Fine detail recovery in blurred images	Superior in fine details, more computationally expensive	Art restoration, digital media
Zhang et al.	2019	ResNet	High efficiency on low resolution images	Pioneer to use ResNet for this purpose	Surveillance, automotive
Johnson & Kim	2022	MobileNet	Optimization for mobile devices	First energy-efficient mobile fitting	Mobile applications, IoT
Patel et al.	2020	Hybrid CNN-GAN	Combination of CNN and GAN for increased accuracy	Better results than using CNN or GAN separately	Professional photography, graphic design
Nguyen et al.	2021	Autoencoder	Efficient in restoring historical images	Less effective in real time compared to CNN	Historical archiving, museums
Harper and Stone	2023	Q-Learning (Reinforcement Learning)	New approach using reinforcement learning	First exploratory study of its kind	Academic research, experimental development





THANK YOU

