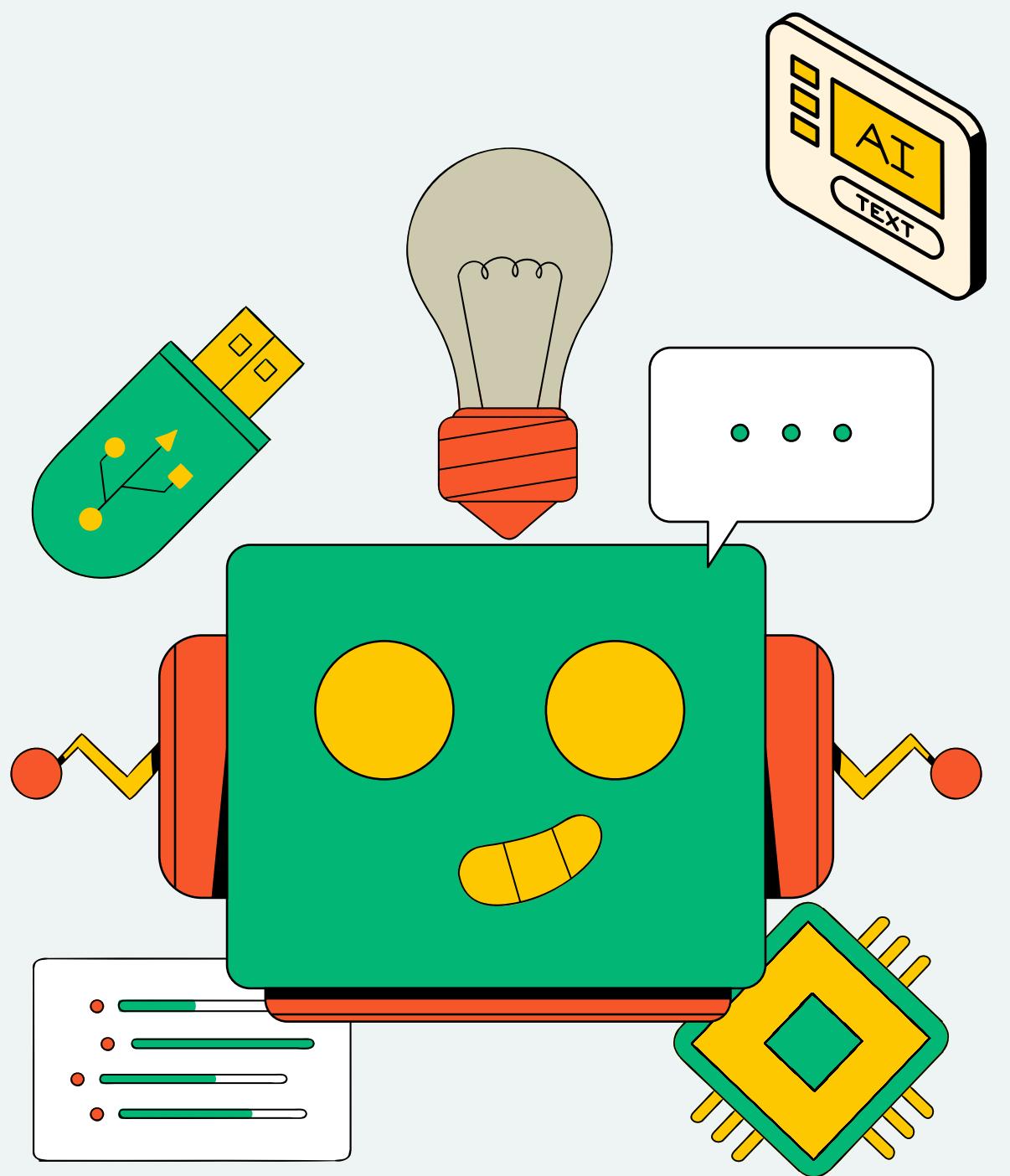


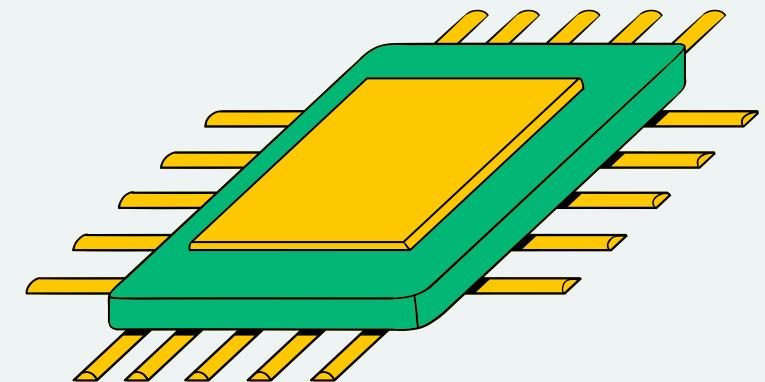
DEEP LEARNING  
PROJECT



# DENOISING DIRTY DOCUMENT IMAGE USING CAE

PRESENTED BY:

RANJIT N



# INTRODUCTION

- Objective: To design and implement a model for removing noise from scanned document images using a deep learning-based autoencoder.
- Problem: Scanned documents often contain various types of noise, making them harder to read or process automatically.
- Solution: We built a convolutional autoencoder to clean noisy document images.



# TRAINING DATA

Noise image: 107.png

There exist several methods to design forms to be filled in. For instance, fields may be surrounded by bounding boxes, by light rectangles or by guiding rulers. These methods specify where to write and, therefore, minimize the effects of skew and overlapping with other parts of the form. These guides can be located on a separate sheet located below the form or they can be printed directly on the form. The use of guides on a separate sheet is much better from the point of view of the quality of the form, but requires giving more instructions and, more importantly, restricts its use to tasks where this type of acquisition is used. Guiding rulers printed on the form are used for this reason. Light rectangles can be removed more easily whenever the handwritten text touches the rulers. Nevertheless, care must be taken into account: The best way to print these light rectangles is to use a high contrast black and white color scheme.

Denoised image: 107.png

There exist several methods to design forms to be filled in. For instance, fields may be surrounded by bounding boxes, by light rectangles or by guiding rulers. These methods specify where to write and, therefore, minimize the effects of skew and overlapping with other parts of the form. These guides can be located on a separate sheet located below the form or they can be printed directly on the form. The use of guides on a separate sheet is much better from the point of view of the quality of the form, but requires giving more instructions and, more importantly, restricts its use to tasks where this type of acquisition is used. Guiding rulers printed on the form are used for this reason. Light rectangles can be removed more easily whenever the handwritten text touches the rulers. Nevertheless, care must be taken into account: The best way to print these light rectangles is to use a high contrast black and white color scheme.

Noise image: 11.png

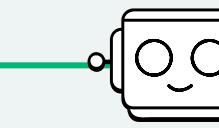
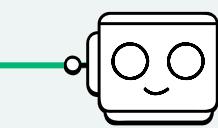
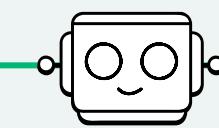
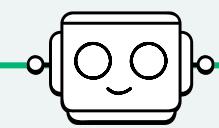
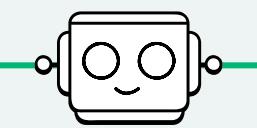
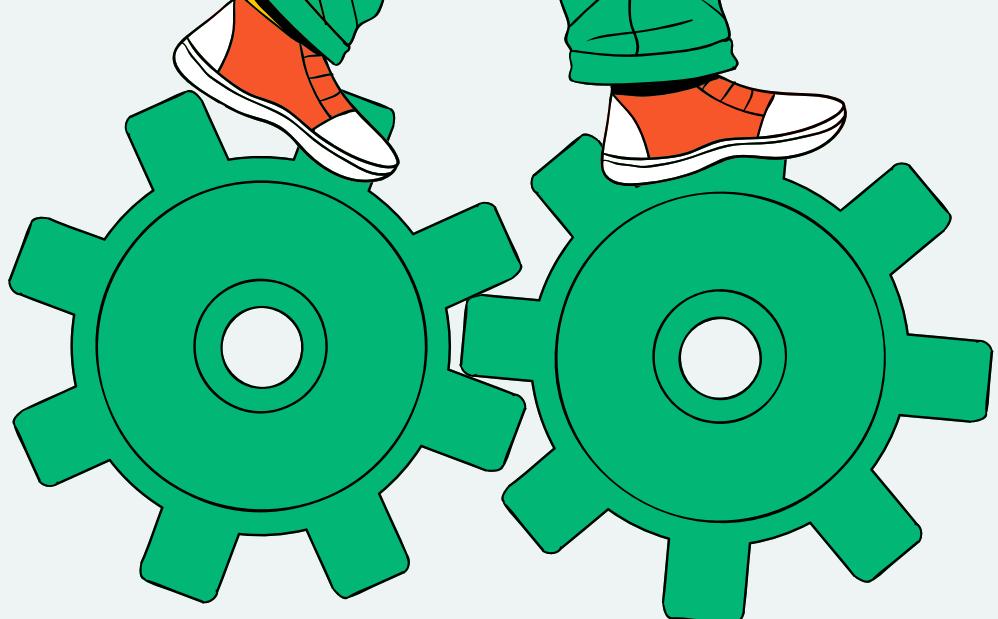
There exist several methods to design forms with fields to be filled in. For instance, fields may be surrounded by bounding boxes, by light rectangles or by guiding rulers. These methods specify where to write and, therefore, minimize the effects of skew and overlapping with other parts of the form. These guides can be located on a separate sheet located below the form or they can be printed directly on the form. The use of guides on a separate sheet is much better from the point of view of the quality of the form, but requires giving more instructions and, more importantly, restricts its use to tasks where this type of acquisition is used. Guiding rulers printed on the form are used for this reason. Light rectangles can be removed more easily whenever the handwritten text touches the rulers. Nevertheless, care must be taken into account: The best way to print these light rectangles is to use a high contrast black and white color scheme.

Denoised image: 11.png

There exist several methods to design forms with fields to be filled in. For instance, fields may be surrounded by bounding boxes, by light rectangles or by guiding rulers. These methods specify where to write and, therefore, minimize the effects of skew and overlapping with other parts of the form. These guides can be located on a separate sheet located below the form or they can be printed directly on the form. The use of guides on a separate sheet is much better from the point of view of the quality of the form, but requires giving more instructions and, more importantly, restricts its use to tasks where this type of acquisition is used. Guiding rulers printed on the form are used for this reason. Light rectangles can be removed more easily whenever the handwritten text touches the rulers. Nevertheless, care must be taken into account: The best way to print these light rectangles is to use a high contrast black and white color scheme.



# PROJECT WORKFLOW



## STEP 1

Data loading and preprocessing.

## STEP 2

Model architecture and training.

## STEP 3

Model evaluation using K-fold cross-validation.

## STEP 4

Training full dataset using convolution auto encoder.

## STEP 5

Performance analysis and optimization.



# EFFICIENCY AND OPTIMIZATION

## CUSTOM LOSS FUNCTION

- Create a loss function combining MSE and SSIM.

## IMPACT

- Improve focus on image reconstruction and perceptual quality.

## IMAGE PREPROCESSING

- Implemented efficient techniques like parallel data loading, resizing, and batching.

## RESULT

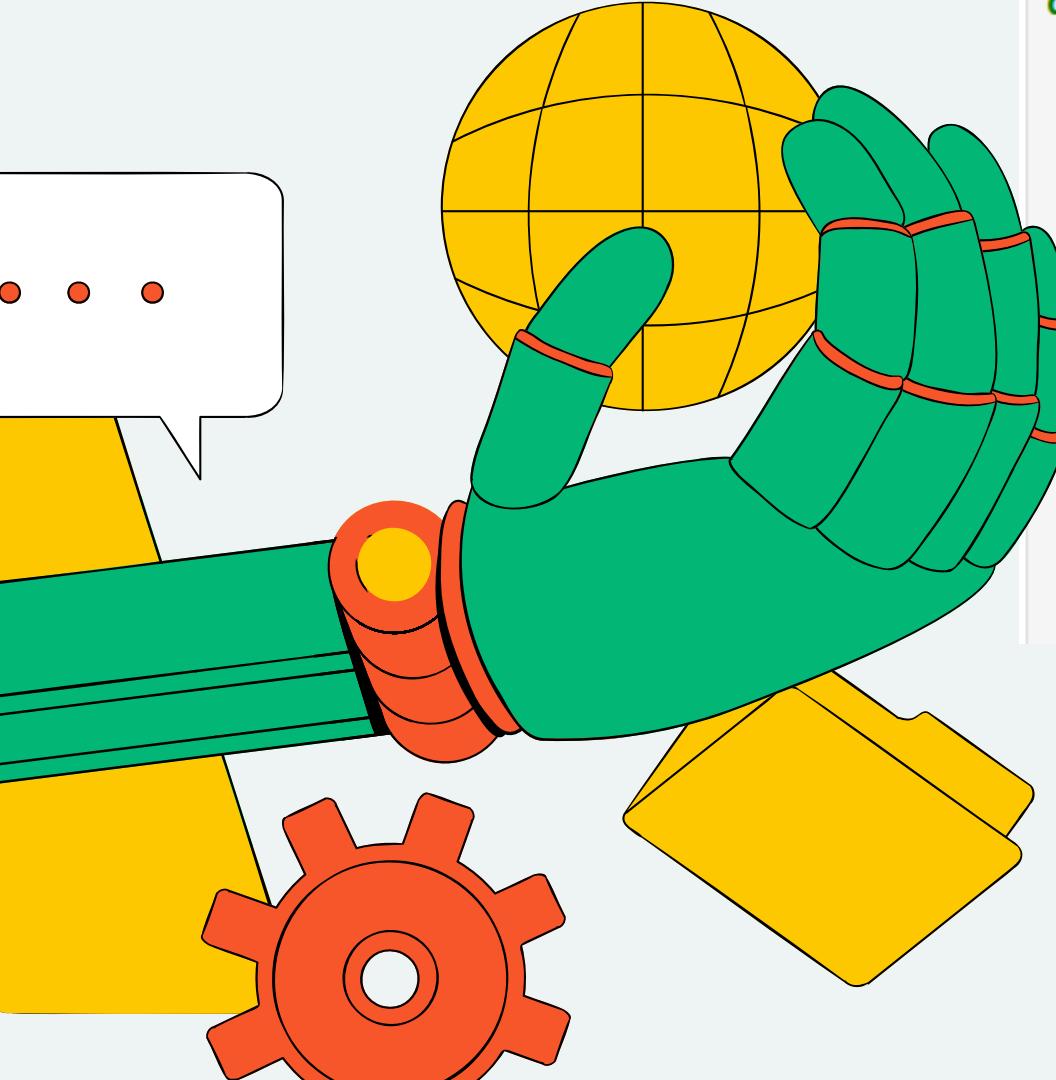
- Faster data handling and reduced training time, boosting productivity.



# CODE



```
def process_image_tensor(image_path, target_size=(270, 350)):  
    img = tf.io.read_file(image_path)  
    img = tf.image.decode_image(img, channels=1)  
    img.set_shape([None, None, 1])  
    img = tf.image.resize(img, target_size)  
    img = img / 255.0  
    return img
```



```
def data_generator_tf(image_paths, clean_paths=None, batch_size=8, target_size=(270, 350)):  
    dataset = tf.data.Dataset.from_tensor_slices(image_paths)  
    dataset = dataset.map(lambda x: process_image_tensor(x, target_size), num_parallel_calls=tf.data.AUTOTUNE)  
  
    if clean_paths is not None:  
        clean_dataset = tf.data.Dataset.from_tensor_slices(clean_paths)  
        clean_dataset = clean_dataset.map(lambda x: process_image_tensor(x, target_size), num_parallel_calls=tf.data.AUTOTUNE)  
        dataset = tf.data.Dataset.zip((dataset, clean_dataset))  
  
    dataset = dataset.batch(batch_size)  
    dataset = dataset.prefetch(buffer_size=tf.data.AUTOTUNE)  
    return dataset
```



# CODE

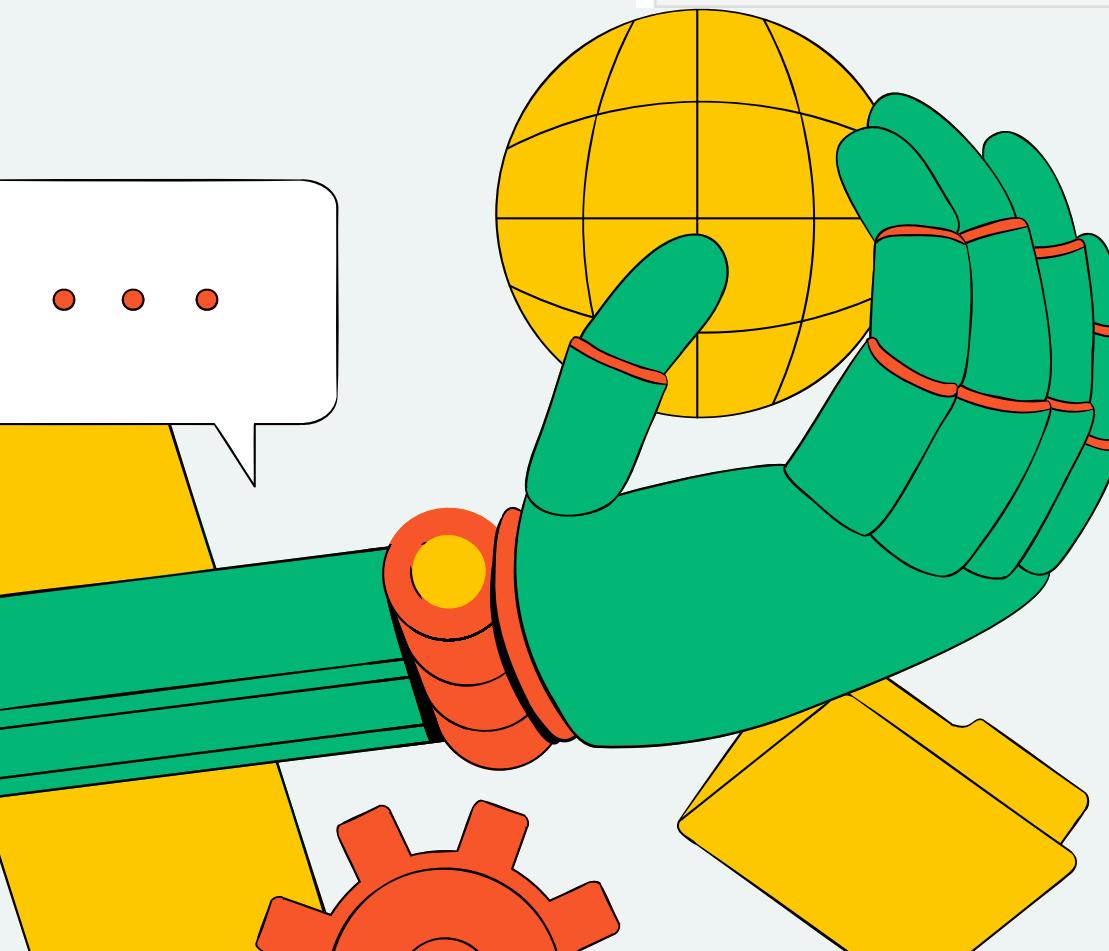


```
def load_data_as_arrays():
    train_paths = sorted([os.path.join(path, 'train', f) for f in os.listdir(os.path.join(path, 'train'))])
    clean_paths = sorted([os.path.join(path, 'train_cleaned', f) for f in os.listdir(os.path.join(path, 'train_cleaned'))])

    train_X = np.array([process_image_tensor(img_path).numpy() for img_path in train_paths])
    train_y = np.array([process_image_tensor(img_path).numpy() for img_path in clean_paths])

    return train_X, train_y

def custom_loss(y_true, y_pred):
    mse = K.mean(K.square(y_true - y_pred))
    ssim_loss = 1 - tf.reduce_mean(tf.image.ssim(y_true, y_pred, max_val=1.0))
    return 0.7 * mse + 0.3 * ssim_loss
```



# K-FOLD CROSS-VALIDATION FOR ROBUSTNESS

01

Divide dataset into 5 parts, trained on 4 and validated on 1.

02

Rotate through all splits.

03

Managed memory limitations by tuning batch sizes.

04

Avoided overfitting through cross-validation.

05

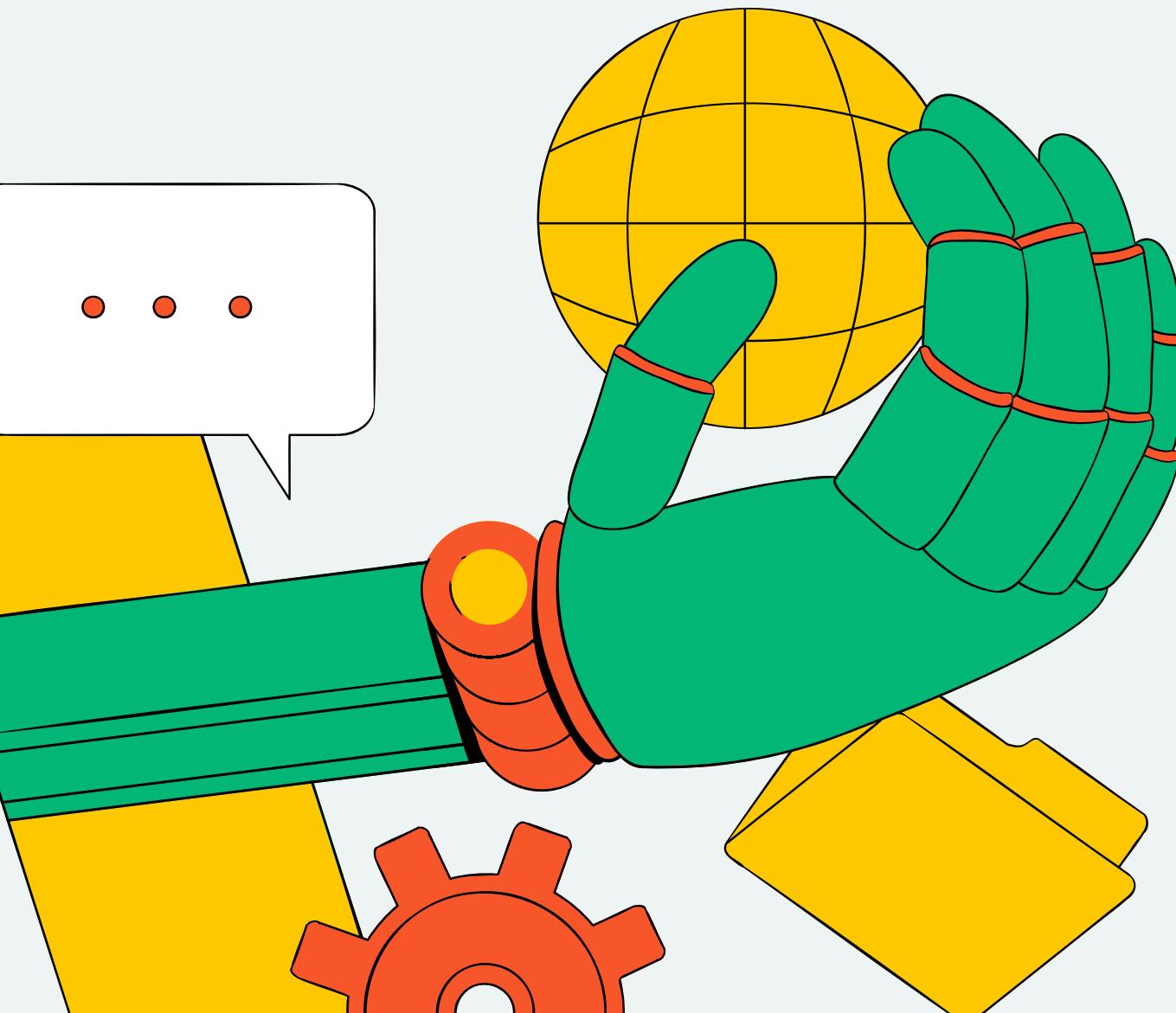
Reliable performance across different data subsets.



# CODE



Average scores across folds:  
Loss: 0.08332  
MAE: 0.06193



```
def train_k_fold(model_builder, x, y, k=5, batch_size=32, epochs=100):
    histories = []
    kf = KFold(n_splits=k, shuffle=True, random_state=42)
    fold_no = 1
    mae_per_fold = []
    loss_per_fold = []

    for train_index, val_index in kf.split(x):
        print(f"\nTraining Fold {fold_no} / {k}")

        X_train, X_val = x[train_index], x[val_index]
        y_train, y_val = y[train_index], y[val_index]

        X_train_resized = tf.image.resize(X_train, [270, 350])
        X_val_resized = tf.image.resize(X_val, [270, 350])

        model = model_builder()

        history = model.fit(X_train_resized, y_train,
                             validation_data=(X_val_resized, y_val),
                             batch_size=batch_size, epochs=epochs, verbose=1)
        histories.append(history.history)
        mae_per_fold.append(history.history['val_mae'][-1])
        loss_per_fold.append(history.history['val_loss'][-1])

        fold_no += 1

    print("\nAverage scores across folds:")
    print(f"Loss: {np.mean(loss_per_fold):.5f}")
    print(f"MAE: {np.mean(mae_per_fold):.5f}")

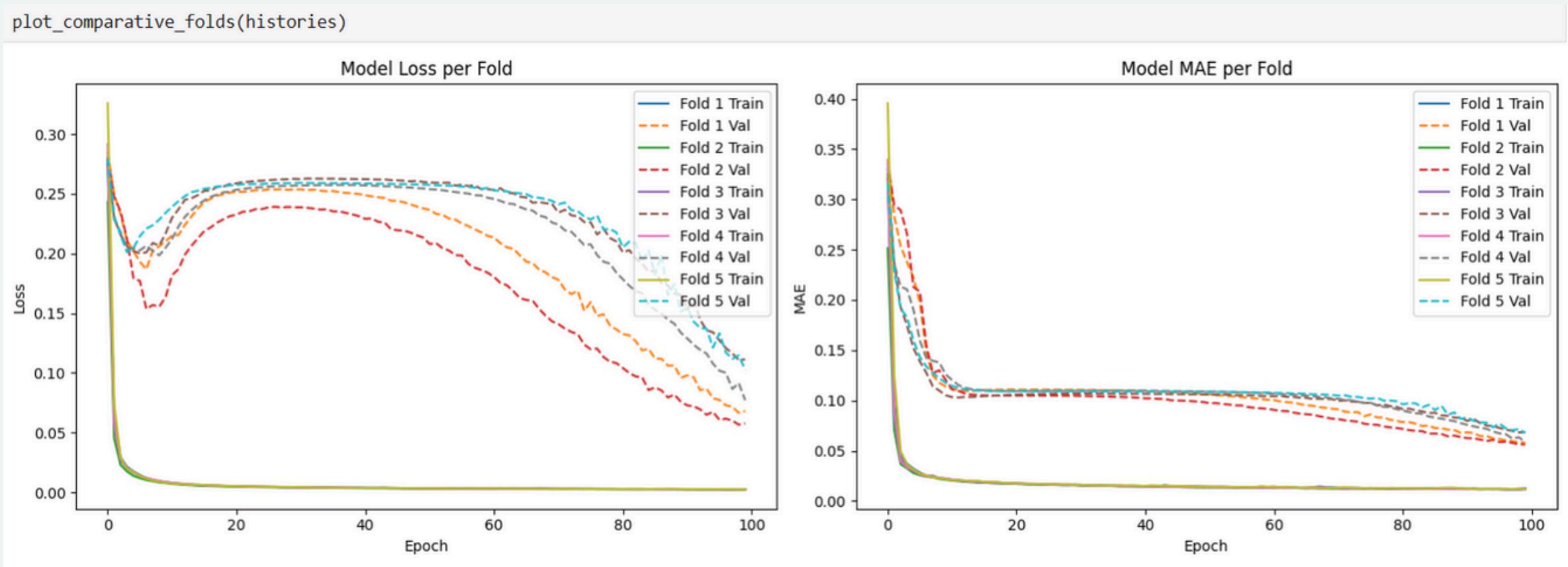
    return model, histories

print("Loading data for K-Fold...")
x, y = load_data_as_arrays()

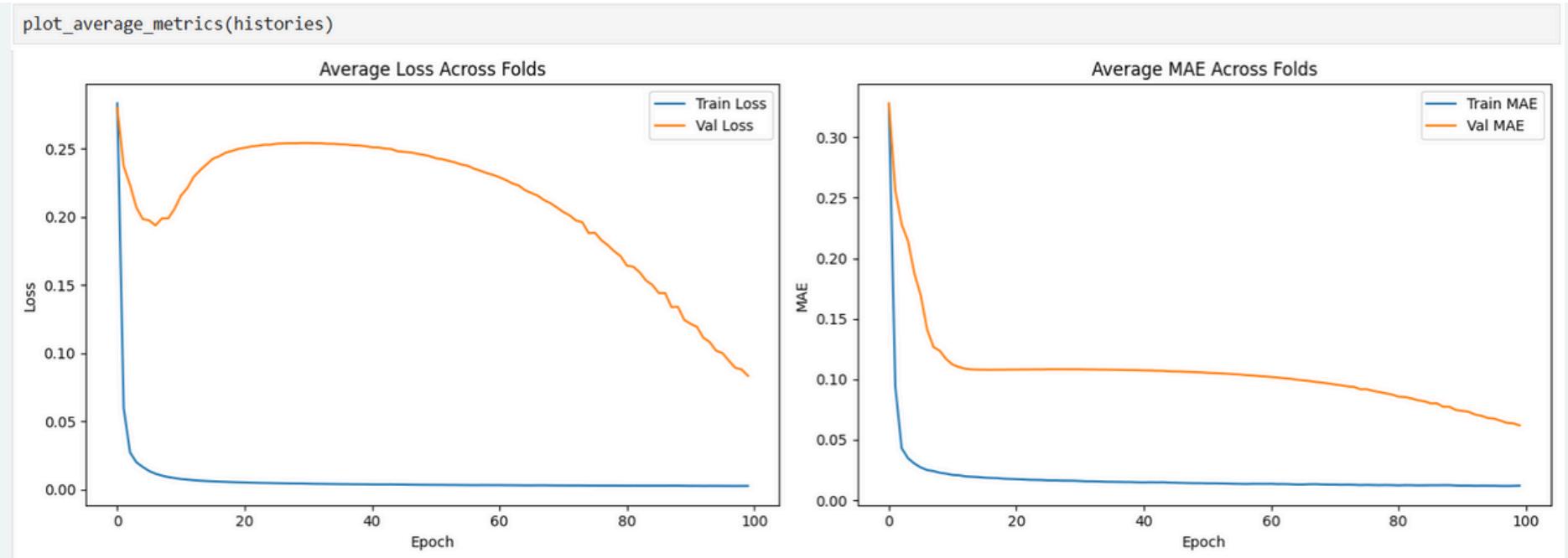
print("Training model with K-Fold Cross-Validation...")
trained_model, histories = train_k_fold(build_model, x, y, k=5, batch_size=32, epochs=100)
```

# VISUALIZATION

```
plot_comparative_folds(histories)
```



```
plot_average_metrics(histories)
```



# MODEL ARCHITECTURE

Model  
Layers

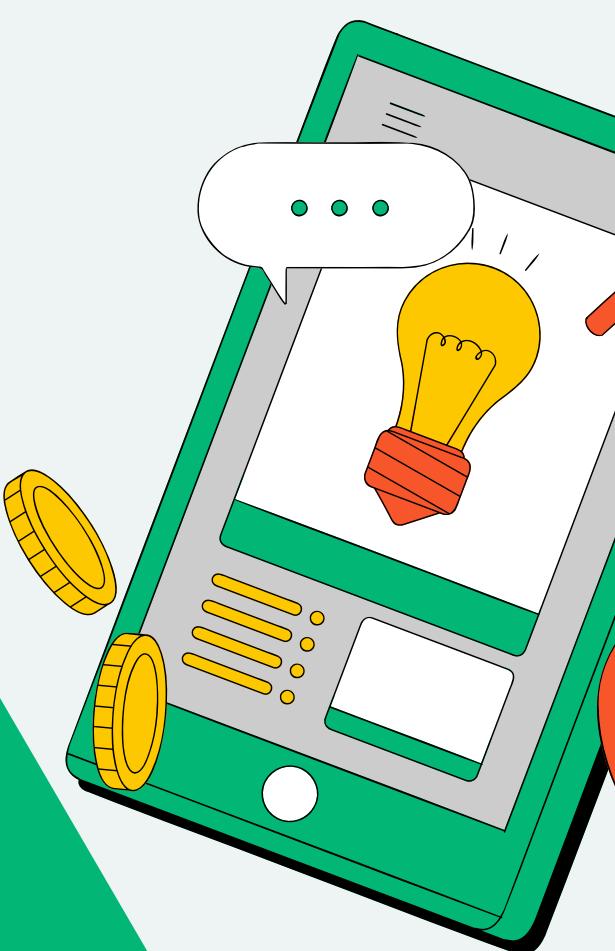
Decoder with  
Conv2D and  
UpSampling

Dropout for  
regularization

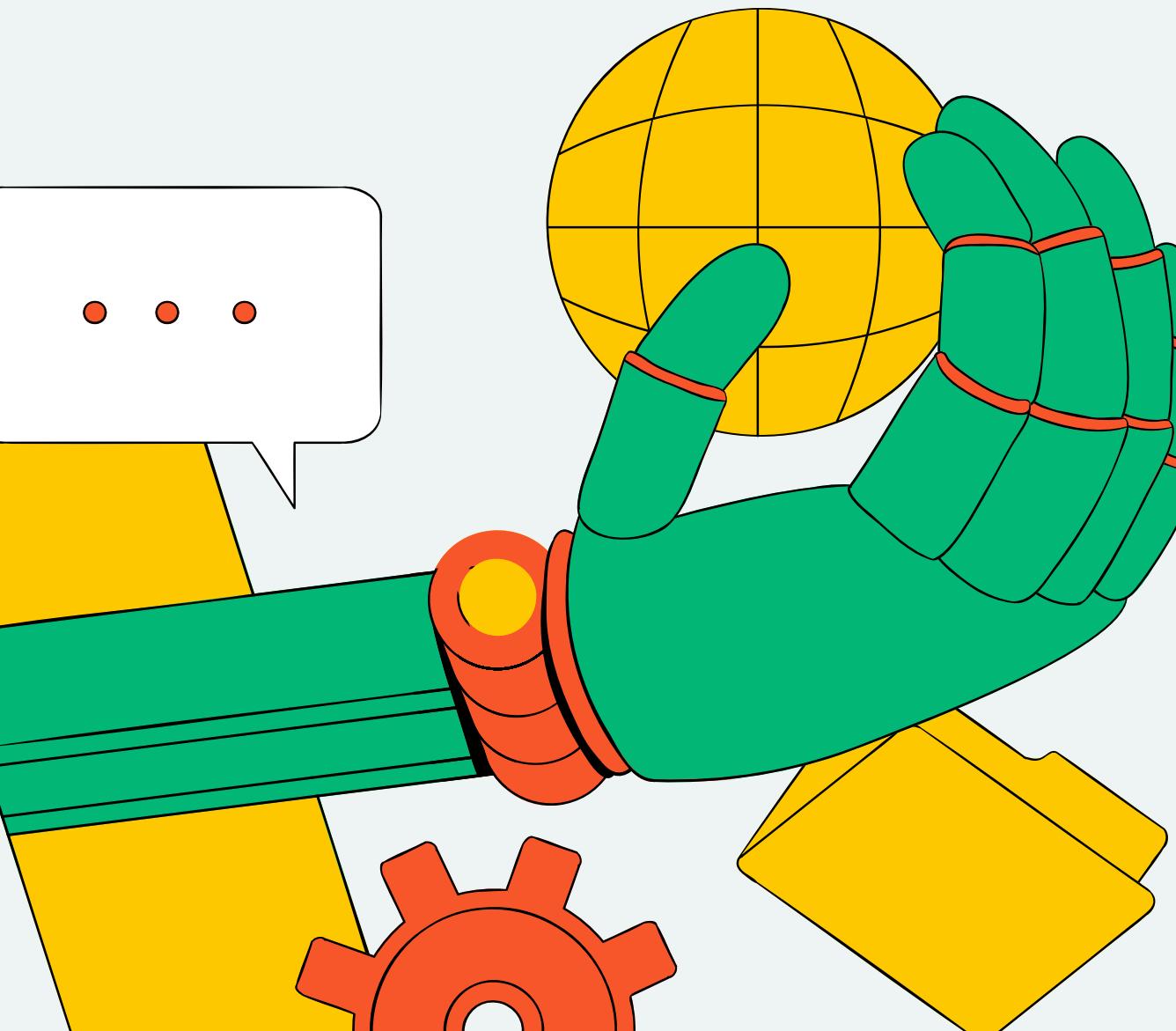
Batch  
Normalization  
for stable  
training

Encoder with  
Conv2D and  
MaxPooling

Mixed  
Precision  
Training:  
Optimized  
memory  
usage on  
GPUs



# CODE



```
def prepare_data(batch_size=32):
    train_paths = sorted([os.path.join(path, 'train', f) for f in os.listdir(os.path.join(path, 'train'))])
    clean_paths = sorted([os.path.join(path, 'train_cleaned', f) for f in os.listdir(os.path.join(path, 'train_cleaned'))])
    train_generator = data_generator_tf(train_paths, clean_paths, batch_size=batch_size)
    return train_generator, len(train_paths)

def build_model(input_shape=(270, 350, 1)):
    input_layer = Input(shape=input_shape)

    # Encoder
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_layer)
    skip1 = x
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = BatchNormalization()(x)
    skip2 = x
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Dropout(0.5)(x)

    # Bridge
    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)

    # Decoder
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = BatchNormalization()(x)
    x = UpSampling2D((2, 2))(x)
    x = Concatenate()([x, skip2])
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    x = Concatenate()([x, skip1])

    output_layer = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

model = Model(inputs=[input_layer], outputs=[output_layer])

optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss=custom_loss, metrics=['mae'])

return model

print("\nTraining final model on the full dataset...")
train_generator, num_train_samples = prepare_data(batch_size=32)
final_model = build_model()

history = final_model.fit(train_generator, epochs=300, verbose=1)
final_model.save('final_model.keras')
```

# IMAGE DENOISING

Original Image 40

A new offline handwritten database for the which contains full Spanish sentences, has re the Spartacus database (which stands for Spar Task of Cursive Script). There were two main this corpus. First of all, most databases do sentences, even though Spanish is a widespread important reason was to create a corpus from tasks. These tasks are commonly used in prac use of linguistic knowledge beyond the lexico process.

As the Spartacus database consisted mainly and did not contain long paragraphs, the writ a set of sentences in fixed places: dedicate the forms. Next figure shows one of the form process. These forms also contain a brief se

Original Image 64

*A new offline handwritten database for the language, which contains full Spanish sentences, has been developed: the Spartacus database (Spanish Restricted-domain Task of Cursive Script). There were two main reasons for creating this. First of all, most databases do not contain Spanish, even though Spanish is a widespread major language. An important reason was to create a corpus for restricted tasks. These tasks are commonly used and allow the use of linguistic knowledge beyond the level in the recognition process.*

Denoised Prediction 40

A new offline handwritten database for the which contains full Spanish sentences, has re the Spartacus database (which stands for Spar Task of Cursive Script). There were two main this corpus. First of all, most databases do sentences, even though Spanish is a widespread important reason was to create a corpus from tasks. These tasks are commonly used in prac use of linguistic knowledge beyond the lexico process.

As the Spartacus database consisted mainly and did not contain long paragraphs, the writ a set of sentences in fixed places: dedicate the forms. Next figure shows one of the form process. These forms also contain a brief se

Denoised Prediction 64

*A new offline handwritten database for the language, which contains full Spanish sentences, has been developed: the Spartacus database (Spanish Restricted-domain Task of Cursive Script). There were two main reasons for creating this. First of all, most databases do not contain Spanish, even though Spanish is a widespread major language. An important reason was to create a corpus for restricted tasks. These tasks are commonly used and allow the use of linguistic knowledge beyond the level in the recognition process.*



# CONCLUSION & FUTURE WORK

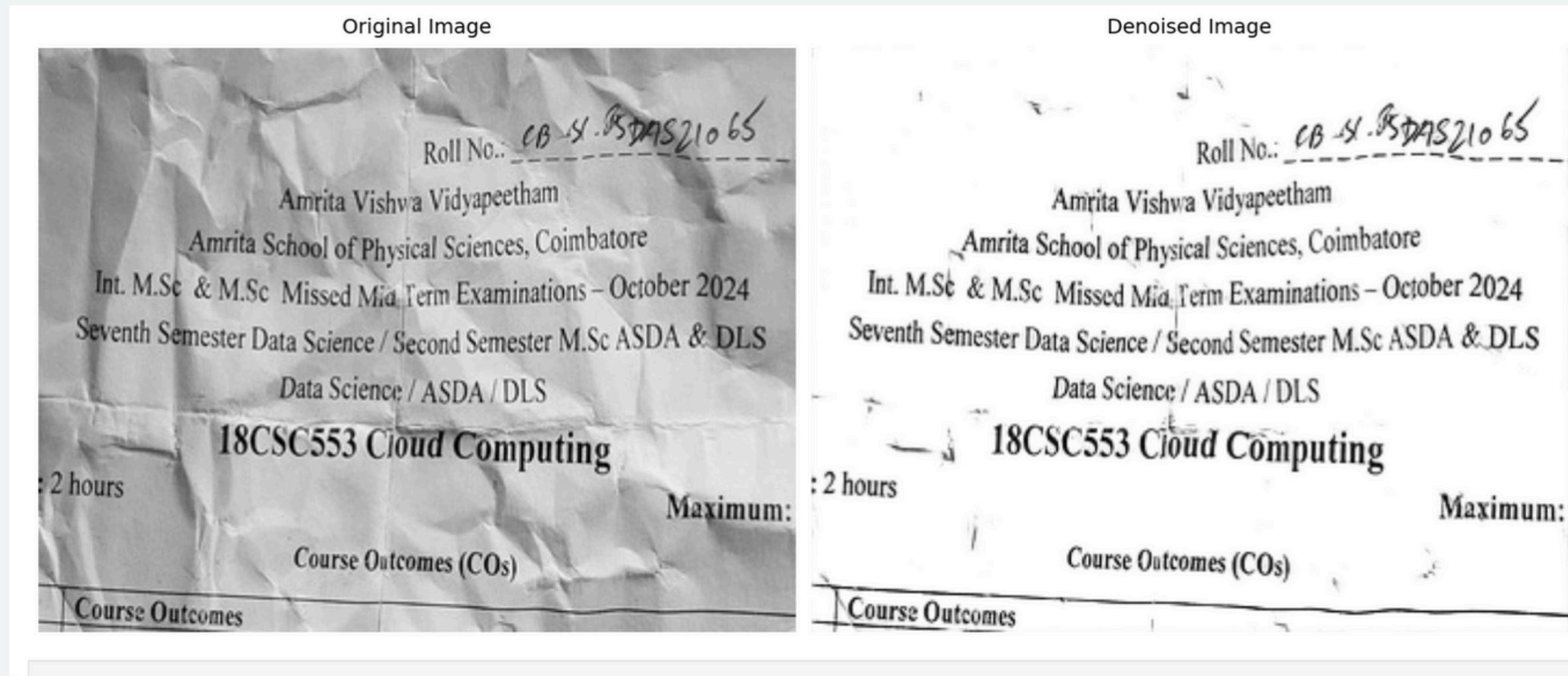
**Summary:** Developed an efficient model for document denoising using a convolutional autoencoder.



- Used techniques like custom loss functions, skip connections, and K-fold cross-validation.
- Future Work: Improve the model to handle more complex noise types. Experiment with additional architectures like UNet



# SAMPLE INPUT IMAGE TO THE MODEL



# THANK YOU

