Representación temporal de alertas ZTF

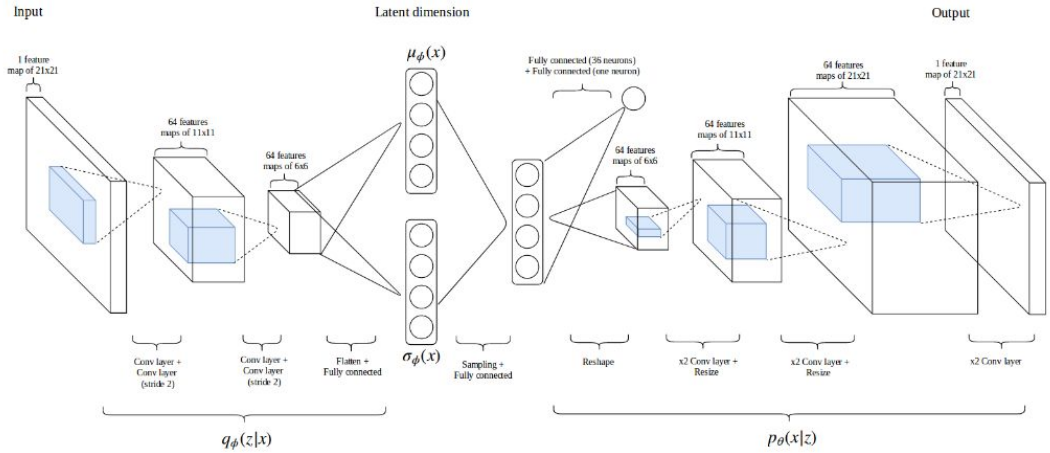
-MP2-

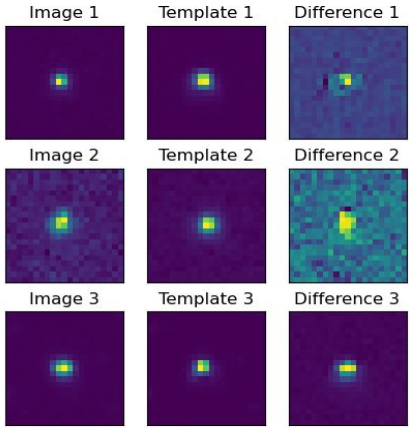
Profesor: Pablo Estévez
Estudiantes: Juan Pablo Contreras
Pascual Marcone
Ayudante: Sebastián Guzmán

Contexto

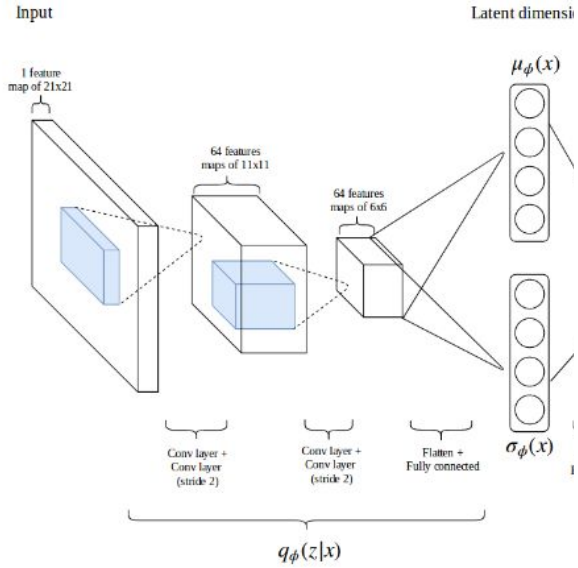
- Replicar AutoEncoder para procesar imágenes, extraer características y agrupar los datos
- Base de datos: Science, Template y Difference
- Input: Difference Image

  Output: Reconstructed Image





Arquitectura del Modelo: Encoder



Encoder: Input imágen 21x21p

- 4 capas convolucionales :
  - 1era y 3era: stride 1
  - 2da y 4ta: stride 2
  - Kernel 3x3 y Feature Maps 64
- 2 capas Fully Connected (FCL)
  - Promedio (μ)
  - Varianza (σ^2)

Implementación Encoder

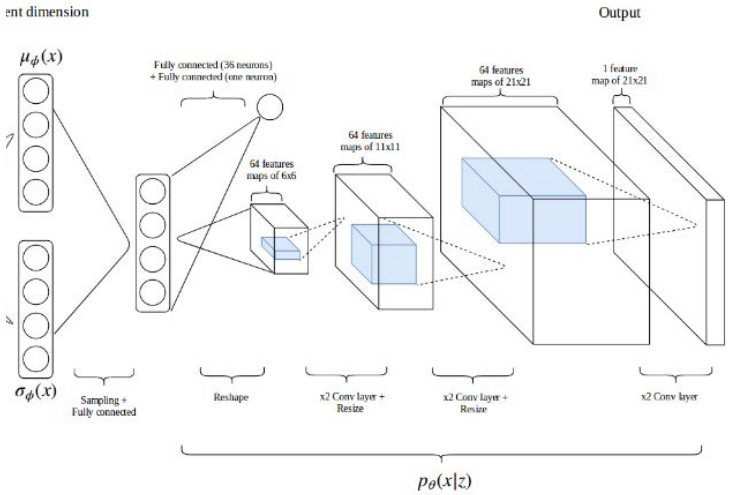
```
self.encoder = nn.Sequential(
      nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1),
      nn.BatchNorm2d(64),
      nn.ReLU(),
      nn.Conv2d(64, 64, kernel_size=3, stride=2, padding=1),
      nn.BatchNorm2d(64),
      nn.ReLU(),
      nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
      nn.BatchNorm2d(64),
      nn.ReLU(),
      nn.Conv2d(64, 64, kernel_size=3, stride=2, padding=1),
      nn.BatchNorm2d(64),
      nn.ReLU(),
      nn.Flatten(),
 )

 self.fc_mu = nn.Linear(64*6*6, latent_dim)
 self.fc_logvar = nn.Linear(64*6*6, latent_dim)
```

Arquitectura del Modelo: Decoder

Decoder: Input variables latentes z

- 1 capa Fully Connected (FCL)
  - Output: 64 Features Maps de 6x6

- 6 capas convolucionales:
  - Stride 1, Kernel 3x3 y Feature Maps 64
  - Nearest neighbor interpolation (2da y 4ta)
  - Aumentan la dimensionalidad hasta 21x21



Implementación Decoder

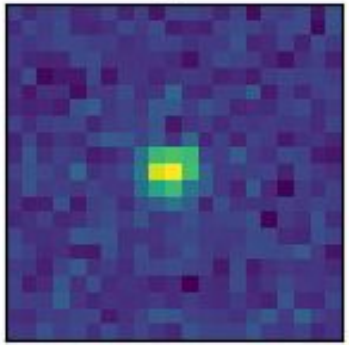
```
self.decoder = nn.Sequential(
      nn.Linear(latent_dim, 64*6*6),
      nn.ReLU(),
      nn.Unflatten(1, (64, 6, 6)),
      nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
      nn.BatchNorm2d(64),
      nn.ReLU(),
      nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
      nn.BatchNorm2d(64),
      nn.ReLU(),
      nn.Upsample(size=(11,11), mode='nearest'),
      nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
      nn.BatchNorm2d(64),
      nn.ReLU(),
      nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
      nn.BatchNorm2d(64),
      nn.ReLU(),
      nn.Upsample(size=(21,21), mode='nearest'),
      nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
      nn.BatchNorm2d(64),
      nn.ReLU(),
      nn.Conv2d(64, 1, kernel_size=3, stride=1, padding=1),
      nn.ReLU()                )
```

Estructura General AutoEncoder

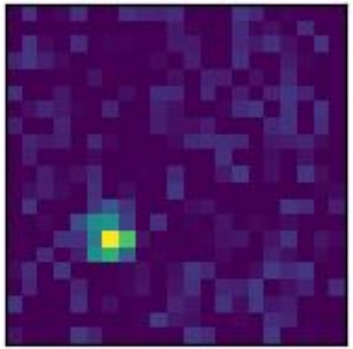
```
class VAE(nn.Module):
    def   init  (self, latent dim):
        super(VAE, self).__init__()

        self.encoder = (...)
        self.decoder = (...)
        self.fc sigma = nn.Sequential(
                nn.Linear(latent_dim, 36),
                nn.ReLU(),
                nn.Linear(36, 1),
                nn.Sigmoid()  # Auto-Regularization)
    def reparametrize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn like(std)
        return mu + eps * std

    def forward(self, x):
        h = self.encoder(x)
        mu = self.fc mu(h)
        logvar = self.fc logvar(h)
        z = self.reparametrize(mu, logvar)
        sigma = self.fc sigma(z)
        reconstruction = self.decoder(z)
        return reconstruction, mu, logvar, sigma
```

Entrenamiento del Modelo

- Utilizando loss_function entre imagen original y reconstruida
- Usando distintos tamaños de datasets
- 6 modelos. 5 entrenados con datos originales, 1 modelo entrenado con datos ficticios de gaussianas descentradas.



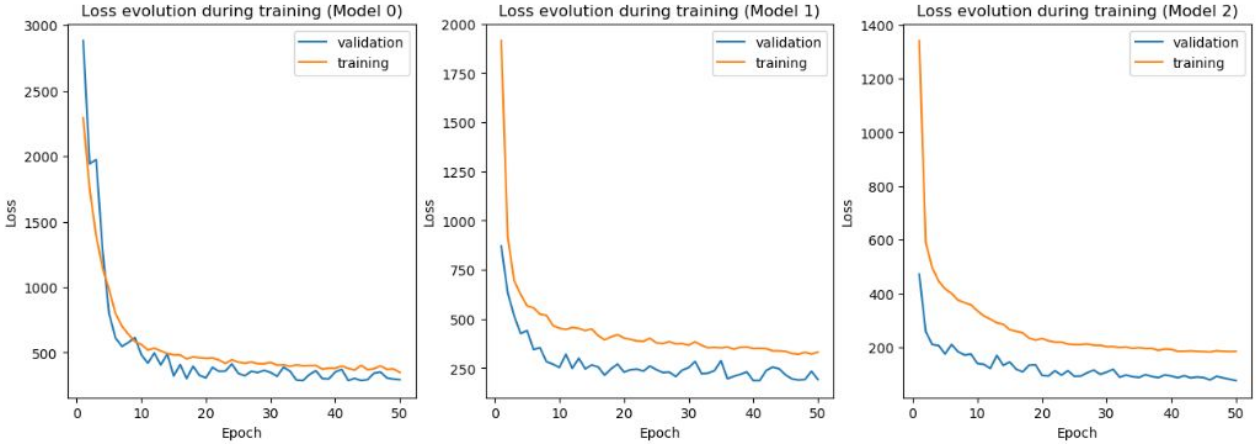
Original



Gaussian

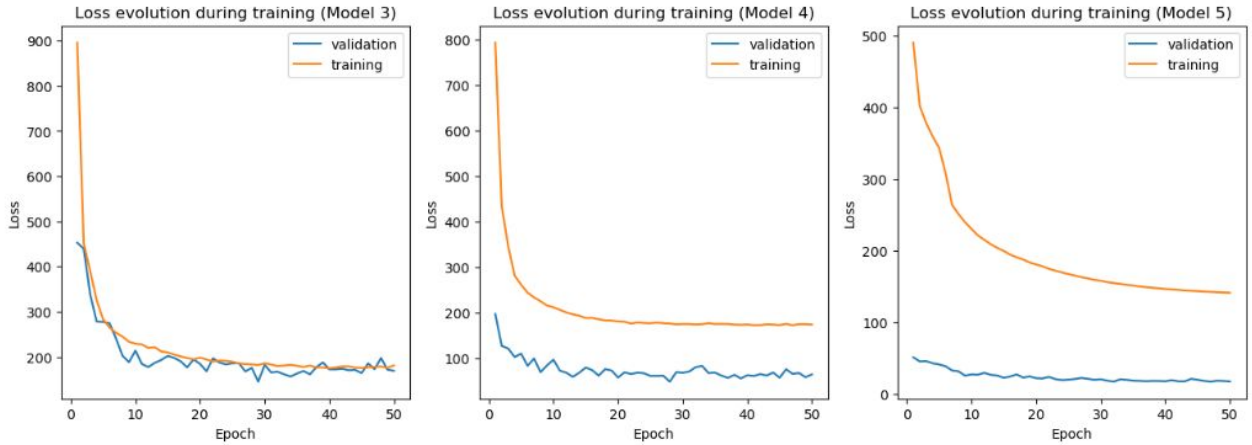
Entrenamiento del Modelo

- Utilizando loss_function entre imagen original y reconstruida
- Usando distintos tamaños de datasets
- 6 modelos. 5 entrenados con datos originales, 1 modelo entrenado con datos ficticios de gaussianas descentradas.

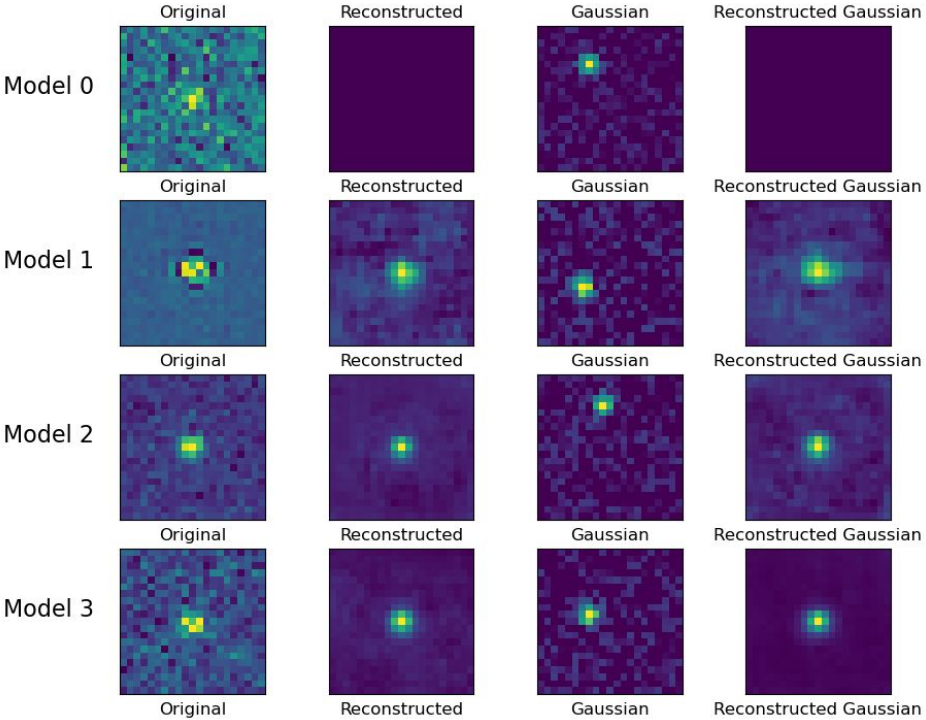


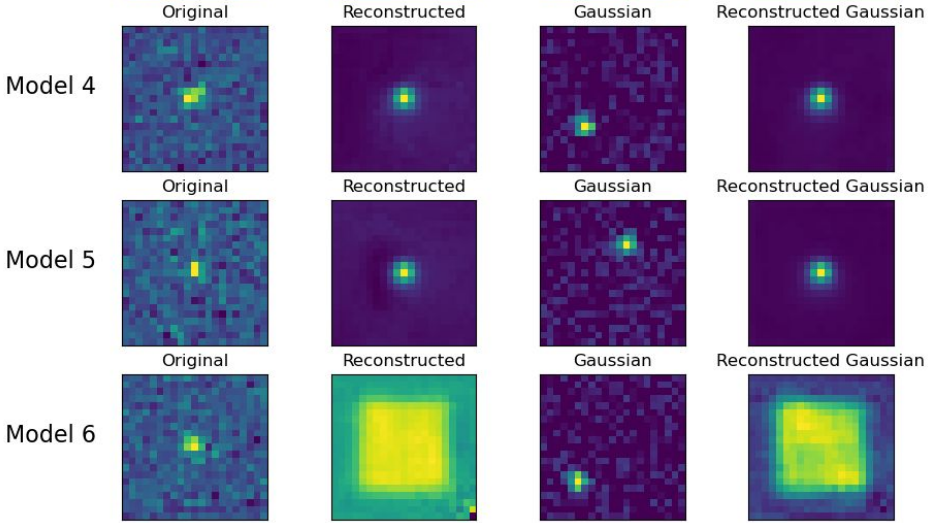
Entrenamiento del Modelo

- Utilizando loss_function entre imagen original y reconstruida
- Usando distintos tamaños de datasets
- 6 modelos. 5 entrenados con datos originales, 1 modelo entrenado con datos ficticios de gaussianas descentradas.



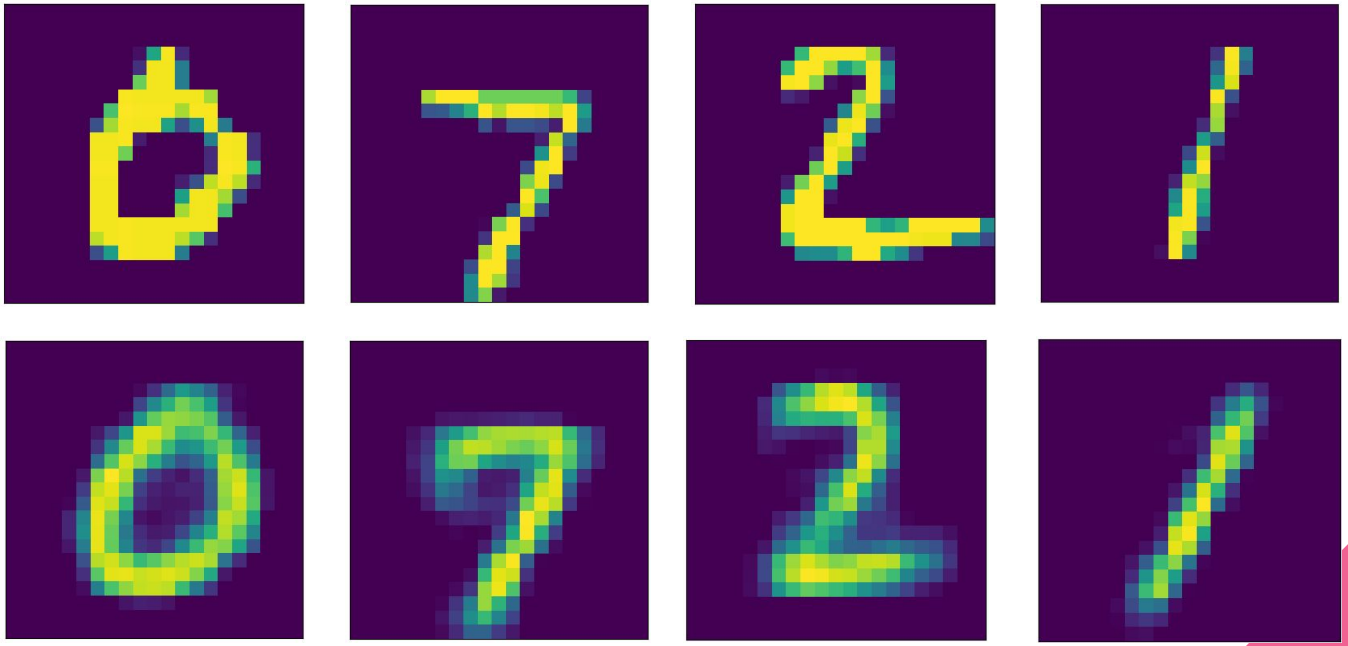
Resultados del modelo





Experimento

Se experimentó con la base de datos MNIST, se obtuvieron resultados interesantes



Siguientes pasos

1. Agregar aumentación al modelo
2. Regular el overfitting
3. Buscar mejores técnicas de entrenamiento
4. Definir mejor las métricas de validación: Loss y Accuracy

Bibliografía

[1] Astorga, N., Huijse, P., Estévez, P. A., Forster, F. (2018, July). Clustering of Astronomical Transient Candidates Using Deep Variational Embedding. In 2018 International Joint Conference on Neural Networks (IJCNN). IEEE

[2] Carrasco-Davis, Rodrigo, et al. "Alert Classification for the ALeRCE Broker System: The Real-time Stamp Classifier." arXiv preprint arXiv:2008.03309 (2020).