```
import torch
import torch.nn as nn
import torch.nn.functional as F

class ANNc(nn.Module):
  def __init__(self, emb_size=64, **kwargs):
    '''CNN based analogy classifier model.

    It generates a value between 0 and 1 (0 for invalid, 1 for valid) based on four input vectors.
    1st layer (convolutional): 128 filters (= kernels) of size h × w = 1 × 2 with strides (1, 2) and relu activation.
    2nd layer (convolutional): 64 filters of size (2, 2) with strides (2, 2) and relu activation.
    3rd layer (dense, equivalent to linear for PyTorch): one output and sigmoid activation.

    Argument:
    emb_size -- the size of the input vectors'''
    super().__init__()
    self.emb_size = emb_size
    self.conv1 = nn.Conv2d(1, 128, (1,2), stride=(1,2))
    self.conv2 = nn.Conv2d(128, 64, (2,2), stride=(2,2))
    self.linear = nn.Linear(64*(emb_size//2), 1)

  def flatten(self, t):
    '''Flattens the input tensor.'''
    t = t.reshape(t.size()[0], -1)
```

```
        return t

    def forward(self, a, b, c, d, p=0):
        """

        Expected input shape:
        - a, b, c, d: [batch_size, emb_size]
        """
        image = torch.stack([a, b, c, d], dim = 2)

        # apply dropout
        if p>0:
            image=F.dropout(image, p)

        x = self.conv1(image.unsqueeze(-3))
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.flatten(x)
        x = self.linear(x)
        output = torch.sigmoid(x)
        return output
```

Link of the dataset on drive :
https://drive.google.com/file/d/1PHsk1CElUQwPSOvvkl6q0GdxWa82tbKa/view?usp=sharing