```python
import torch as torch
import torch.nn as nn


import torch
import torch.nn as nn
import numpy as np


class MLP(nn.Module):
    def __init__(self, input_size:int, action_size:int,
hidden_size:int=256,non_linear:nn.Module=nn.ReLU):
        """
        input: tuple[int]
            The input size of the image, of shape (channels, height, width)
        action_size: int
            The number of possible actions
        hidden_size: int
            The number of neurons in the hidden layer

        This is a seperate class because it may be useful for the bonus questions
        """
        super(MLP, self).__init__()
        # ========== YOUR CODE HERE ==========
        # TODO:
        # self.linear1 =
        # self.output =
        # self.non_linear =
        # ====================================

        self.linear1 = nn.Linear(input_size, hidden_size)
        self.output = nn.Linear(hidden_size, action_size)
        self.non_linear = non_linear()

        # ========== YOUR CODE ENDS ==========

    def forward(self, x:torch.Tensor)->torch.Tensor:
        # ========== YOUR CODE HERE ==========

        x = self.non_linear(self.linear1(x))
        x = self.output(x)

        # ========== YOUR CODE ENDS ==========
        return x

class Nature_Paper_Conv(nn.Module):
    """
    A class that defines a neural network with the following architecture:
    - 1 convolutional layer with 32 8x8 kernels with a stride of 4x4 w/ ReLU activation
    - 1 convolutional layer with 64 4x4 kernels with a stride of 2x2 w/ ReLU activation
    - 1 convolutional layer with 64 3x3 kernels with a stride of 1x1 w/ ReLU activation
    - 1 fully connected layer with 512 neurons and ReLU activation.
    Based on 2015 paper 'Human-level control through deep reinforcement learning' by Mnih et al
    """
    def __init__(self, input_size:tuple[int], action_size:int,**kwargs):
        """
        input: tuple[int]
            The input size of the image, of shape (channels, height, width)
        action_size: int
            The number of possible actions
        **kwargs: dict
            additional kwargs to pass for stuff like dropout, etc if you would want to
implement it
        """
        super(Nature_Paper_Conv, self).__init__()
        # ========== YOUR CODE HERE ==========
```

```python
        c, h, w = input_size

        self.CNN = nn.Sequential(
            nn.Conv2d(c, 32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1),
            nn.ReLU()
        )

        # Compute output size after conv layers
        def conv2d_out(size, kernel, stride):
            return (size - kernel) // stride + 1

        convw = conv2d_out(conv2d_out(conv2d_out(w, 8, 4), 4, 2), 3, 1)
        convh = conv2d_out(conv2d_out(conv2d_out(h, 8, 4), 4, 2), 3, 1)
        linear_input_size = convw * convh * 64

        # Must match weight file: 512 hidden units
        self.MLP = MLP(input_size=linear_input_size, action_size=action_size, hidden_size=512)


        # ========== YOUR CODE ENDS ==========

    def forward(self, x:torch.Tensor)->torch.Tensor:
        # ========== YOUR CODE HERE ==========

        x = self.CNN(x)
        x = x.view(x.size(0), -1)
        x = self.MLP(x)

        # ========== YOUR CODE ENDS ==========
        return x
```