Báo cáo tìm hiểu các models

(09/01/2023 – 13/01/2023)

I. Dữ liệu

1. Giới thiệu: Cifar10

Bộ dữ liệu CIFAR-10 (gồm 10 classes) là một tập hợp con của bộ dữ liệu Tiny Images và bao gồm 60.000 hình ảnh màu 32x32. Các hình ảnh được gắn nhãn với một trong 10 loại loại khác nhau: máy bay, ô tô (nhưng không phải xe tải hoặc xe bán tải), chim, mèo, nai, chó, ếch, ngựa, tàu và xe tải (nhưng không phải xe bán tải). Có 6.000 hình ảnh mỗi lớp với 5.000 hình ảnh training và 1.000 hình ảnh testing mỗi lớp.

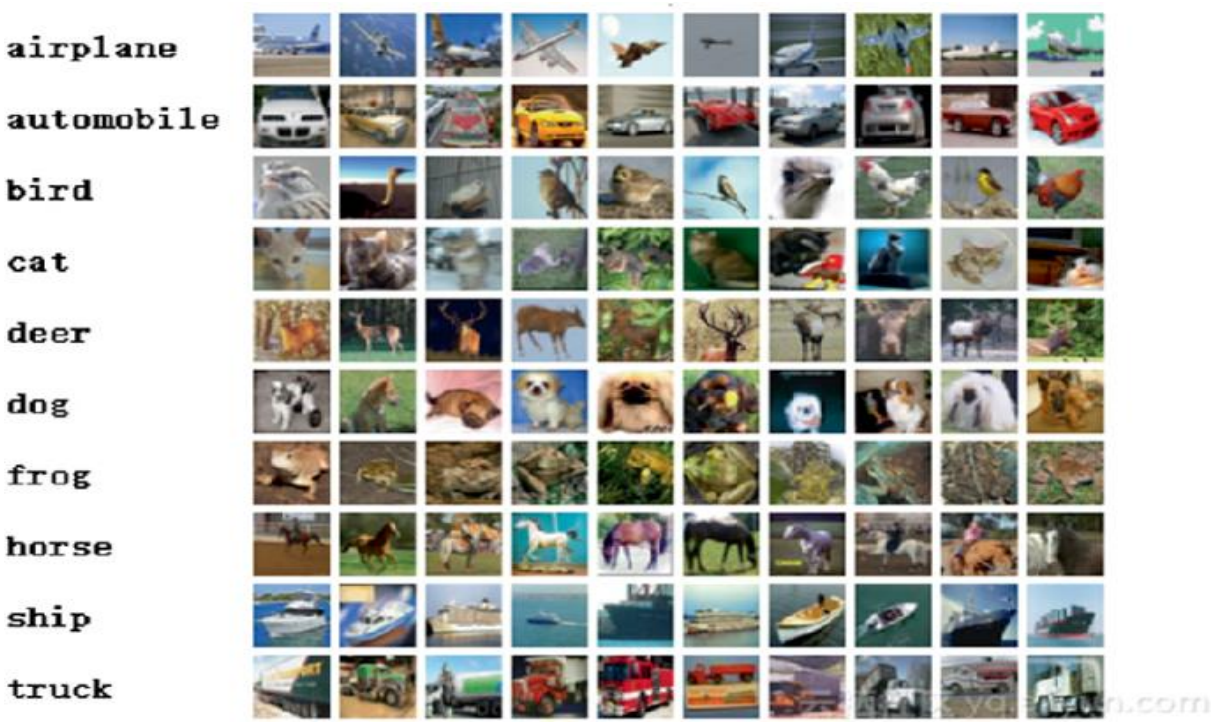
Sử dụng bằng cách import thư viện:

a. **Pytorch:** torchvision.datasets.CIFAR10

b. **Tensorflow:** tfds.image_classification.Cifar10

- **Keras:** tf.keras.datasets.cifar10.load_data()

Một số hình ảnh có trong tập dataset:



2. **Tiền xử lý dữ liệu:**

   a. **Tải dữ liệu để training:**

```
# Tải dữ liệu
(X_train, y_train),(X_test, y_test) = datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 13s 0us/step
```

   b. **Đặt tên các classes có trong dữ liệu:** Các classes trong bộ dữ liệu đang chỉ được đặt tên là số từ 0 đến 9

```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

   c. **Chuẩn hoá dữ liệu:** Cần sử lý dữ liệu trước khi đưa vào model để training

```
# Chuẩn hoá dữ liệu
X_train = X_train/225
X_test = X_test/255
y_train = y_train.flatten()
y_test = y_test.flatten()
```

II. Models

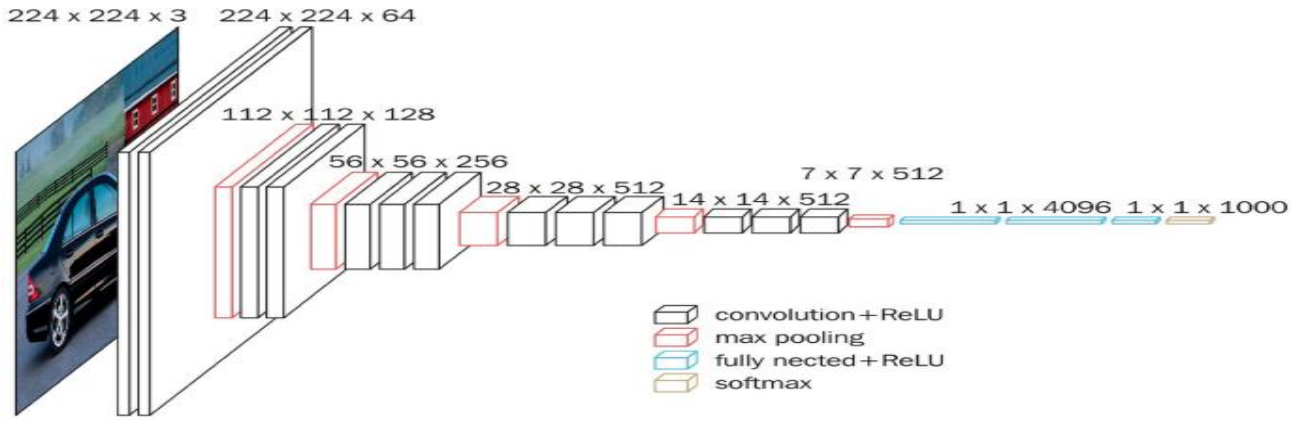
1. VGG16:

a. Giới thiệu:

VGG16 là mạng CNN(Convolutional Neural Networks) dùng trong phân loại hình ảnh được đề xuất bởi K. Simonyan and A. Zisserman, University of Oxford. Model sau khi train bởi mạng VGG16 đạt độ chính xác 92.7% top-5 test trong dữ liệu ImageNet gồm 14 triệu hình ảnh thuộc 1000 lớp khác nhau.

Size ảnh input của VGG16 là RGB(224x224), RGB là mô hình màu có 3 màu chủ đạo là đỏ(red), lục(green), xanh dương(blue) nên VGG16 nhận ma trận đầu vào (224,224,3).

VGG16 có 16 layers, bao gồm 13 layers conv (2 layers conv - conv và 3 layers conv - conv - conv) đều có kernel 3x3, sau mỗi layers conv là các layers maxpooling giảm kích thước ảnh xuống 0.5 và cuối cùng là 3 layers full connection.



Sau khi nhận ảnh đầu vào 224x224x3.

Hai lớp conv đầu tiên, mỗi lớp có 64 kernel kích thước 3x3. Sau đó, một lớp maxpooling 2x2 thì cho ra output ảnh là 112x112x64.

Hai lớp conv tiếp theo, mỗi lớp có 128 kernel kích thước 3x3 và một lớp maxpooling 2x2 cho ra output ảnh là 56x56x128.

Ba lớp conv đầu tiên, mỗi lớp có 256 kernel có kích thước 3x3 và sau đó là một lớp maxpooling 2x2 cho ra output ảnh là 28x28x256.

Ba lớp conv tiếp theo, mỗi lớp có 512 kernel có kích thước 3x3 và lớp maxpooling 2x2 cho ra output 14x14x512.

Ba lớp conv cuối cùng, mỗi lớp có 512 kernel có kích thước 3x3. Sau đó là một lớp maxpooling 2x2 cho ra output ảnh có kích thước 7x7x512.

Cuối cùng là 3 lớp Dense, với 2 lớp đầu fully - connected có 4096 nút và lớp fully - connected cuối cùng có 1000 nút và cho ra output là phân loại của ảnh đầu vào.

b. Ứng dụng: Code demo

Xây dựng model với 6 block trong đó có block 1 đến block 5 và classification block

```python
# Xây dựng models
VGG16 = models.Sequential([
    # Layer input
    layers.Input(shape=X_train[0].shape),

    # Block 1
    # Có 2 lớp Convolution2D có 64 filters trích xuất các feature của ảnh
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),

    # Block 2
    # Có 2 lớp Convolution2D có 128 filters
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),

    # Block 3
    # Có 3 lớp Convolution2D có 256 filters
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),

    # Block 4
    # Có 3 lớp Convolution2D có 512 filters
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),
```

```
    # Block 5
    # Có 3 lớp Convolution2D có 512 filters
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),

    # Classification block
    # Bao gồm 1 lớp làm phẳng Flatten và 3 lớp Dense để phân loại hình ảnh
    layers.Flatten(),
    layers.Dense(4096, activation='relu'),
    layers.Dense(4096, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

Xong khi xây dựng model, ta compile model để biên dịch model với thuật toán tối ưu là adam, hàm mất mát là sparse categorical crossentropy để model không cần thực hiện one-hot encoding với metrics là độ chính xác.

```
[ ]  # Compile model
     VGG16.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

Training model với 50 epochs (50 lần train lại tập dữ liệu liên tục)

```
# Train model
history = VGG16.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50)
```

```
Epoch 22/50
1563/1563 [==============================] - 53s 34ms/step - loss: 2.3028 - accuracy: 0.0996 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 23/50
1563/1563 [==============================] - 53s 34ms/step - loss: 2.3028 - accuracy: 0.0971 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 24/50
1563/1563 [==============================] - 53s 34ms/step - loss: 2.3028 - accuracy: 0.0999 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 25/50
1563/1563 [==============================] - 53s 34ms/step - loss: 2.3028 - accuracy: 0.0996 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 26/50
1563/1563 [==============================] - 53s 34ms/step - loss: 2.3028 - accuracy: 0.0967 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 27/50
1563/1563 [==============================] - 53s 34ms/step - loss: 2.3028 - accuracy: 0.0956 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 28/50
1563/1563 [==============================] - 53s 34ms/step - loss: 2.3028 - accuracy: 0.0977 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 29/50
1563/1563 [==============================] - 53s 34ms/step - loss: 2.3028 - accuracy: 0.0975 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 30/50
```

Đánh giá model cho accuracy là 1%

```
# Đánh giá model
VGG16.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 3s 10ms/step - loss: 2.3026 - accuracy: 0.1000
[2.3026318550109863, 0.10000000149011612]
```

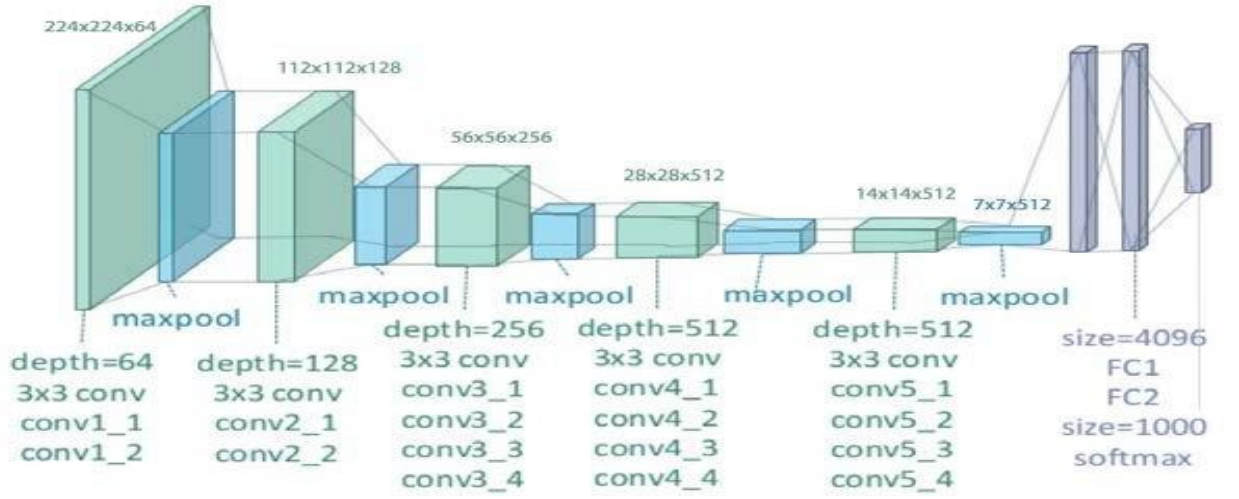
2. VGG19:

a. Giới thiệu:

VGG19 là mô hình mạng CNN nâng cấp của VGG16 được dùng trong phân loại hình ảnh.

Size ảnh input của VGG19 là RGB(224x224), RGB là mô hình màu có 3 màu chủ đạo là đỏ(red), lục(green), xanh dương(blue) nên VGG19 nhận ma trận đầu vào (224,224,3).

VGG19 có 19 layers, bao gồm 16 layers conv (2 layers conv - conv và 3 layers conv - conv - conv - conv) đều có kernel 3x3, sau mỗi layers conv là các layers maxpooling giảm kích thước ảnh xuống 0.5 và cuối cùng là 3 layers full connection.



Sau khi nhận ảnh đầu vào 224x224x3.

Hai lớp conv đầu tiên, mỗi lớp có 64 kernel kích thước 3x3. Sau đó, một lớp maxpooling 2x2 thì cho ra output ảnh là 112x112x64.

Hai lớp conv tiếp theo, mỗi lớp có 128 kernel kích thước 3x3 và một lớp maxpooling 2x2 cho ra output ảnh là 56x56x128.

Bốn lớp conv đầu tiên, mỗi lớp có 256 kernel có kích thước 3x3 và sau đó là một lớp maxpooling 2x2 cho ra output ảnh là 28x28x256.

Bốn lớp conv tiếp theo, mỗi lớp có 512 kernel có kích thước 3x3 và lớp maxpooling 2x2 cho ra output 14x14x512.

Bốn lớp conv cuối cùng, mỗi lớp có 512 kernel có kích thước 3x3. Sau đó là một lớp maxpooling 2x2 cho ra output ảnh có kích thước 7x7x512.

Cuối cùng là 3 lớp Dense, với 2 lớp đầu fully - connected có 4096 nút và lớp fully - connected cuối cùng có 1000 nút và cho ra output là phân loại của ảnh đầu vào.

b. Ứng dụng: Code demo

Xây dựng model với 6 block trong đó có block 1 đến block 5 và classification block.

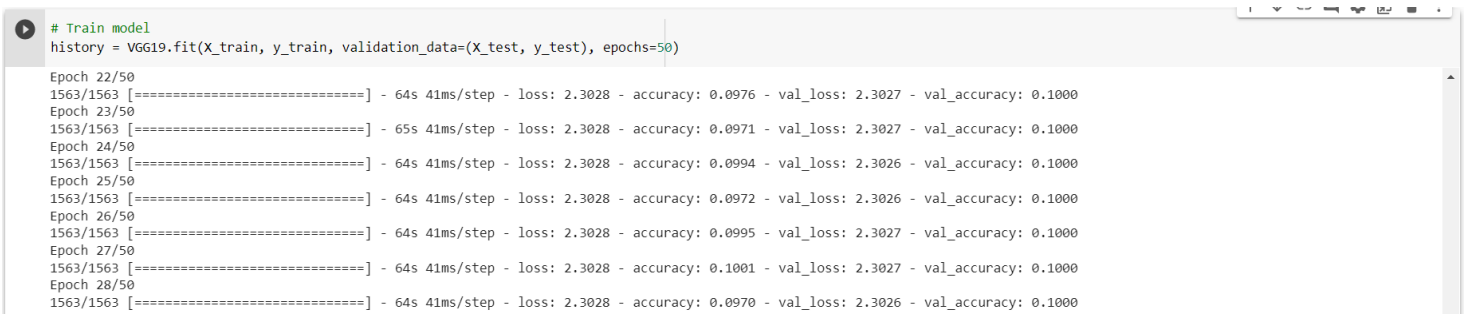
```
# Xây dựng models
VGG19 = models.Sequential([
    # Layer input
    layers.Input(shape=X_train[0].shape),

    # Block 1
    # Có 2 lớp Convolution2D có 64 filters trích xuất các feature của ảnh
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),

    # Block 2
    # Có 2 lớp Convolution2D có 128 filters
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),

    # Block 3
    # Có 4 lớp Convolution2D có 256 filters
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),
```

```
    # Block 4
    # Có 4 lớp Convolution2D có 512 filters
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),

    # Block 5
    # Có 4 lớp Convolution2D có 512 filters
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),

    # Classification block
    # Bao gồm 1 lớp làm phẳng Flatten và 3 lớp Dense để phân loại hình ảnh
    layers.Flatten(),
    layers.Dense(4096, activation='relu'),
    layers.Dense(4096, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

Compile model để biên dịch model với thuật toán tối ưu là adam, hàm mất mát là sparse categorical crossentropy để model không cần thực hiện one-hot encoding với metrics là độ chính xác.

```
# Compile model
VGG19.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

Training model với 50 epochs (50 lần train lại tập dữ liệu liên tục).



Đánh giá model cho accuracy là 1%.

```
[ ] # Đánh giá model
    VGG19.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 4s 12ms/step - loss: 2.3026 - accuracy: 0.1000
[2.302643299102783, 0.10000000149011612]
```

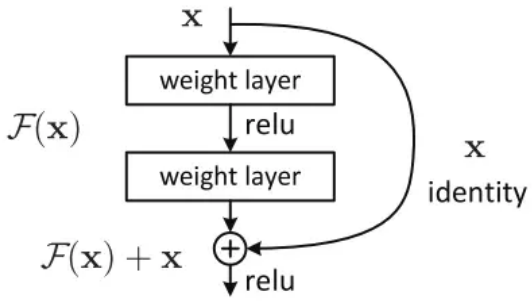
3. ResNet50:

a. Giới thiệu:

ResNet (Residual Network) được giới thiệu đến công chúng vào năm 2015 và thậm chí đã giành được vị trí thứ 1 trong cuộc thi ILSVRC 2015 với tỉ lệ lỗi top 5 chỉ 3.57%. Không những thế nó còn đứng vị trí đầu tiên trong cuộc thi ILSVRC and COCO 2015 với ImageNet Detection, ImageNet localization, Coco detection và Coco segmentation. Hiện tại thì có rất nhiều biến thể của kiến trúc ResNet với số lớp khác nhau như ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152,...Với tên là ResNet theo sau là một số chỉ kiến trúc ResNet với số lớp nhất định.

Mạng ResNet (R) là một mạng CNN được thiết kế để làm việc với hàng trăm lớp. Một vấn đề xảy ra khi xây dựng mạng CNN với nhiều lớp chập sẽ xảy ra hiện tượng Vanishing Gradient dẫn tới quá trình học tập không tốt.

Cho nên giải pháp mà ResNet đưa ra là sử dụng kết nối "tắt" đồng nhất để xuyên qua một hay nhiều lớp. Một khối như vậy được gọi là một Residual Block, như trong hình sau:

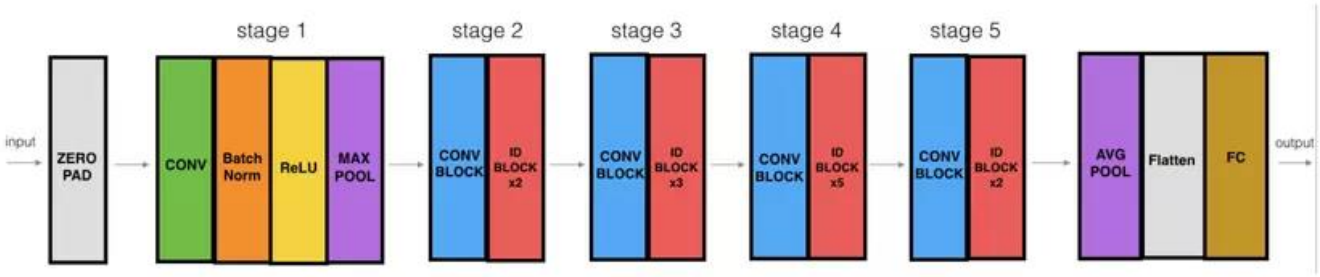


ResNet gần như tương tự với các mạng gồm có convolution, pooling, activation và fully-connected layer. Ảnh bên trên hiển thị khối dư được sử dụng trong mạng. Xuất hiện một mũi tên cong xuất phát từ đầu và kết thúc tại cuối khối dư. Hay nói cách khác là sẽ bổ sung Input X vào đầu ra của layer, hay chính là phép cộng mà ta thấy trong hình minh họa, việc này sẽ chống lại việc đạo hàm bằng 0, do vẫn còn cộng thêm X. Với H(x) là giá trị dự đoán, F(x) là giá trị thật (nhãn), chúng ta muốn H(x) bằng hoặc xấp xỉ F(x). Việc F(x) có được từ x như sau:

$$X \rightarrow \text{weight1} \rightarrow \text{ReLU} \rightarrow \text{weight2}$$

Giá trị H(x) có được bằng cách:

$$F(x) + x \rightarrow \text{ReLU}$$



Zero - padding : Input với (3,3)

Stage 1: Tích chập (Conv1) với 64 filters với shape (7,7), sử dụng stride (2,2). BatchNorm, MaxPooling (3,3).

Stage 2: Convolutiontal block sử dụng 3 filter với size 64x64x256, f=3, s=1. Có 2 Identity blocks với filter size 64x64x256, f=3.

Stage 3: Convolutional sử dụng 3 filter size 128x128x512, f=3, s=2. Có 3 Identity blocks với filter size 128x128x512, f=3.

Stage 4: Convolutional sử dụng 3 filter size 256x256x1024, f=3, s=2. Có 5 Identity blocks với filter size 256x256x1024, f=3.

Stage 5: Convolutional sử dụng 3 filter size 512x512x2048, f=3, s=2. Có 2 Identity blocks với filter size 512x512x2048, f=3.

The 2D Average Pooling: sử dụng với kích thước (2,2).

The Flatten.

Fully Connected (Dense): sử dụng softmax activation.

b. Ứng dụng: Code demo

Xây dựng khối residual có trong model.

```
# Build residual block

class ResidualBlock(Layer):
    def __init__(self, num_channels, output_channels, strides=1, is_identity=True, **kwargs):
        super(ResidualBlock, self).__init__(**kwargs)
        self.is_identity = is_identity
        self.conv1 = Conv2D(num_channels, padding='same', kernel_size=3, strides=1)
        self.bn = BatchNormalization()
        self.conv2 = Conv2D(num_channels, padding='same', kernel_size=3)

        if not self.is_identity:
            self.conv3 = Conv2D(num_channels, kernel_size=1, strides=1)
        else:
            self.conv3 = None
        self.conv4 = Conv2D(output_channels, padding='same', kernel_size=1, strides=strides)
        self.relu = ReLU()

    def call(self, X):
        # Init residual
        Y = X
        if self.conv3 is not None:
            Y = self.conv3(X)
        # Conv block 1
        X = self.conv1(X)
        X = self.bn(X)
        X = self.relu(X)
        # Conv block 2
        X = self.conv2(X)
        X = self.bn(X)
        # Add residual to output of stacked
        X += Y
        # Last conv block
        X = self.conv4(X)

        return self.relu(X)
```

Xây dựng model.

```
# Build our ResNet model

class ResNetModel(tf.keras.Model):
    def __init__(self, residual_blocks, nb_classes):
        super(ResNetModel, self).__init__()
        self.conv1 = Conv2D(kernel_size=7, filters=64, strides=2, padding='same')
        self.bn = BatchNormalization()
        self.max_pool = MaxPool2D(pool_size=(3, 3), strides=2, padding='same')
        self.relu = ReLU()
        self.residual_blocks = residual_blocks
        self.gap = GlobalAvgPool2D()
        self.fc = Dense(units=nb_classes)

    def call(self, X):
        # First layer before residual blocks
        X = self.conv1(X)
        X = self.bn(X)
        X = self.relu(X)
        X = self.max_pool(X)
        # Residual blocks
        for block in residual_blocks:
            X = block(X)
        # After residual block
        X = self.gap(X)
        # Fully connected layer
        X = self.fc(X)

        return X
```

Thêm các thông số cho khối residual khác nhau có trong model.

```
[ ] # Define residual blocks

    residual_blocks = [
        # Two residual block with identity mapping
        ResidualBlock(num_channels=64, output_channels=64, strides=2, is_identity=True),
        ResidualBlock(num_channels=64, output_channels=64, strides=2, is_identity=True),
        # Next 3 [conv mapping + identity mapping]
        ResidualBlock(num_channels=64, output_channels=128, strides=2, is_identity=False),
        ResidualBlock(num_channels=128, output_channels=128, strides=2, is_identity=True),
        ResidualBlock(num_channels=128, output_channels=256, strides=2, is_identity=False),
        ResidualBlock(num_channels=256, output_channels=256, strides=2, is_identity=True),
        ResidualBlock(num_channels=256, output_channels=512, strides=2, is_identity=False),
        ResidualBlock(num_channels=512, output_channels=512, strides=2, is_identity=True)
    ]
```

Tạo model với 10 classes của tập dataset.

```
# Tạo model

model = ResNetModel(residual_blocks=residual_blocks, nb_classes=10)
model.build(input_shape=(None, 32, 32, 3))
```

Compile model để biên dịch model với thuật toán tối ưu là adam, hàm mất mát là sparse categorical crossentropy để model không cần thực hiện one-hot encoding với metrics là độ chính xác.

```
[ ]  # Compile model
     model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

Training model với 50 epochs (50 lần train lại tập dữ liệu liên tục).

```
# Train model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50)

Epoch 22/50
1563/1563 [==============================] - 29s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 23/50
1563/1563 [==============================] - 28s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 24/50
1563/1563 [==============================] - 28s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 25/50
1563/1563 [==============================] - 28s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 26/50
1563/1563 [==============================] - 29s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 27/50
1563/1563 [==============================] - 29s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 28/50
1563/1563 [==============================] - 28s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 29/50
1563/1563 [==============================] - 28s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 30/50
1563/1563 [==============================] - 28s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 31/50
1563/1563 [==============================] - 28s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 32/50
1563/1563 [==============================] - 28s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 33/50
1563/1563 [==============================] - 28s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 34/50
1563/1563 [==============================] - 28s 18ms/step - loss: 3.8090 - accuracy: 0.1000 - val_loss: 3.8090 - val_accuracy: 0.1000
Epoch 35/50
```

Đánh giá model với accuracy là 1%.

```
[ ]  # Đánh giá model
     model.evaluate(X_test,y_test)

     313/313 [==============================] - 2s 7ms/step - loss: 3.8090 - accuracy: 0.1000
     [3.809035539627075, 0.10000000149011612]
```

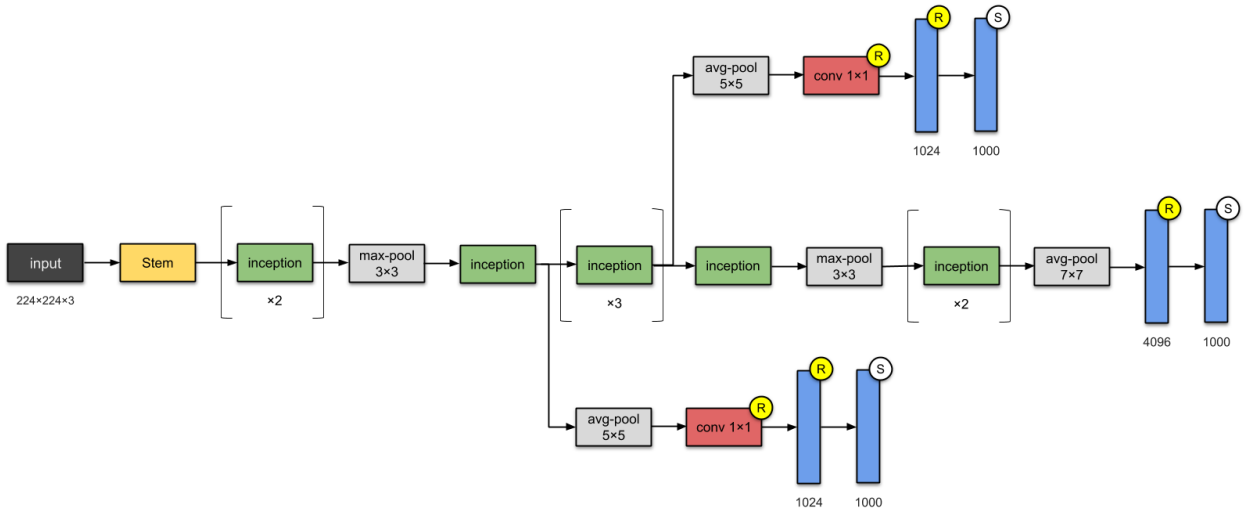
4. Inception:

a. Giới thiệu:

Inception ra đời vào năm 2014 bởi Google.Inception Module được sử dụng trong CNN để cho phép tính toán hiệu quả hơn và tạo ra nhiều Networks sâu hơn thông qua phép giảm chiều với các convolutions có kết cấu 1×1 xếp chồng lên nhau. Module này được thiết kế để giải quyết vấn đề về chi phí tính toán, cũng như overfitting, trong số các vấn đề khác. Tóm lại, giải pháp là sử dụng nhiều bộ lọc kernel có kích cỡ khác nhau bên trong CNN, và thay vì xếp chồng chúng lên nhau theo tuần tự, ta xếp chúng sao cho chúng có thể vận hành trên cùng một lớp.

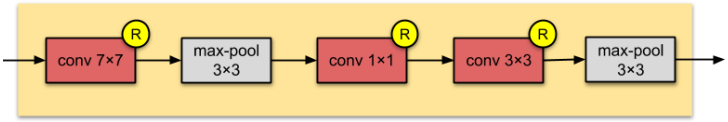
Size ảnh input của Inception là RGB(224x224), RGB là mô hình màu có 3 màu chủ đạo là đỏ(red), lục(green), xanh dương(blue) nên ResNet nhận ma trận đầu vào (224,224,3).

Inception Network có 22 layers khi chỉ tính các layers có parameters (hoặc 27 layers nếu ta tính thêm lớp pooling). Số lượng layers tổng cộng (các blocks building độc lập nhau) được dung để xây dựng network này là khoảng 100. Tuy nhiên, con số này phụ thuộc vào hệ cơ sở hạ tầng học máy đang được sử dụng.

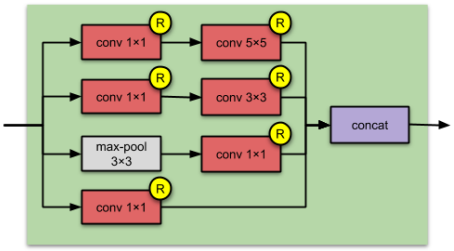


Điểm quan trọng trong cấu trúc từng lớp của model Inception là khối Inception

❖ Khối Inception:



Stem



Inception module

Khối Inception sẽ bao gồm 4 nhánh song song. Các bộ lọc kích thước lần lượt là 1×1, 3×3, 5×5 được áp dụng trong Inception Module giúp trích lọc được đa dạng đặc trưng trên những vùng nhận thức có kích thước khác nhau:

+ Block 1: gồm các bộ lọc có kích thước kernel 1×1.

+ Block 2: gồm các bộ lọc có kích thước kernel 1×1, tiếp theo sau đó là các bộ lọc có kích thước kernel 3×3(3×3 reduce).

+ Block 3: gồm các bộ lọc có kích thước kernel 1×1, tiếp theo sau đó là các bộ lọc có kích thước kernel 5×5(5×5 reduce).

+ Block 4: gồm lớp max-pooling có kích thước kernel 3×3, tiếp theo sau đó là bộ locj có kích thước kernel 1×1.

Ở đầu các nhánh 1, 2, 4 từ trên xuống, phép tích chập 1×1 được sử dụng trên từng điểm ảnh như một kết nối fully connected nhằm mục đích giảm độ sâu kênh và số lượng tham số của mô hình. Ví dụ: Ở block trước chúng ta có kích thước width x height x channels = 12 × 12 × 256. Sau khi áp dụng 32 bộ lọc kích thước 1x1 sẽ không làm thay đổi width, height và độ sâu giảm xuống 32, output shape lúc này có kích thước là 12 × 12 × 32. Ở layer liền sau, khi thực hiện tích chập trên toàn bộ độ sâu, chúng ta chỉ khởi tạo các bộ lọc có độ sâu 32 thay vì 256. Do đó đó số lượng tham số giảm đi một cách đáng kể.

Nhánh thứ 3 từ trên xuống giúp chúng ta giảm chiều dữ liệu bằng một layer max-pooling kích thước 3x3 và sau đó áp dụng bộ lọc kích thước 1x1 để thay đổi số kênh.

Các nhánh áp dụng padding và stride sao cho đầu ra có cùng kích cỡ chiều dài và chiều rộng. Cuối cùng ta concatenate toàn bộ kết quả đầu ra của các khối theo kênh để thu được output có kích thước bằng với input.

b. Ứng dụng: Không hoàn thành!

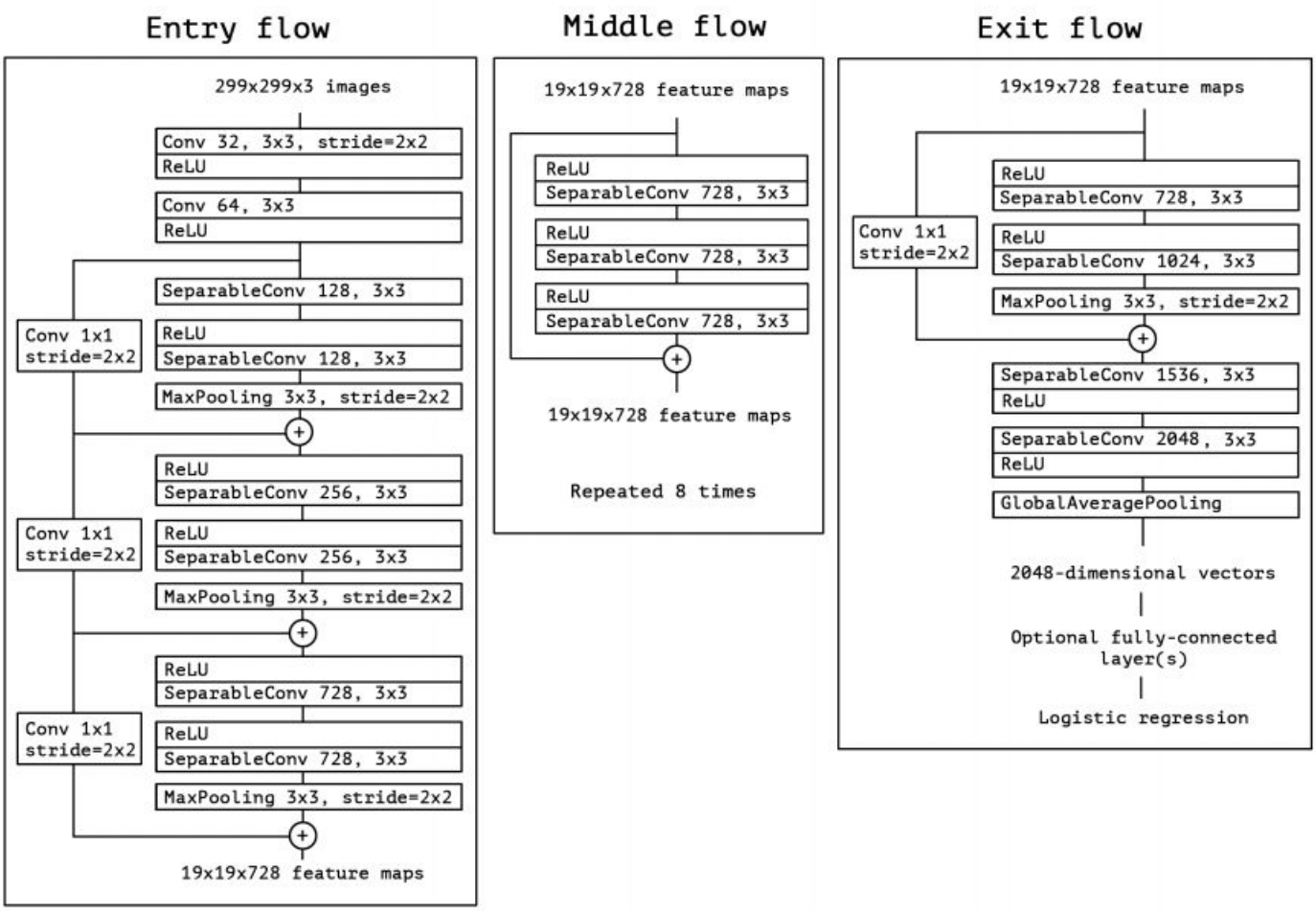
5. Xception:

a. Giới thiệu:

Xception là một kiến trúc CNN có thể phân tách theo chiều sâu (Depthwise Separable Convolutions). Một tích chập có thể phân tách theo chiều sâu có thể được hiểu là mô-đun Inception với số lượng tháp cực lớn. Kiến trúc mạng thần kinh tích chập sâu mới lấy cảm hứng từ Inception, trong đó các mô-đun Inception đã được thay thế bằng các cấu trúc tích chập có thể phân tách theo chiều sâu.

Size ảnh input của Xception là RGB(224x224), RGB là mô hình màu có 3 màu chủ đạo là đỏ(red), lục(green), xanh dương(blue) nên ResNet nhận ma trận đầu vào (224,224,3).

Đầu tiên dữ liệu sẽ qua entry flow, tiếp theo qua Middle flow 8 lần, cuối cùng đi qua exit flow. Chú ý tất cả lớp Convolution và lớp SeparableConvolution được chuẩn hóa Batch Normalization.



b. Ứng dụng: [Code demo]

Xây dựng 3 hàm entry flow, middle flow và exit flow

```
[ ]  def entry_flow(inputs):

        x = layers.Conv2D(32, 3, strides=2, padding='same')(inputs)
        x = layers.BatchNormalization()(x)
        x = layers.Activation('relu')(x)

        x = layers.Conv2D(64, 3, padding='same')(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation('relu')(x)

        previous_block_activation = x  # Set aside residual

        # Blocks 1, 2, 3 are identical apart from the feature depth.
        for size in [128, 256, 728]:
            x = layers.Activation('relu')(x)
            x = layers.SeparableConv2D(size, 3, padding='same')(x)
            x = layers.BatchNormalization()(x)

            x = layers.Activation('relu')(x)
            x = layers.SeparableConv2D(size, 3, padding='same')(x)
            x = layers.BatchNormalization()(x)

            x = layers.MaxPooling2D(3, strides=2, padding='same')(x)

            residual = layers.Conv2D(  # Project residual
                size, 1, strides=2, padding='same')(previous_block_activation)
            x = layers.add([x, residual])  # Add back residual
            previous_block_activation = x  # Set aside next residual

        return x
```

```
[ ]
    def middle_flow(x, num_blocks=8):

        previous_block_activation = x

        for _ in range(num_blocks):
            x = layers.Activation('relu')(x)
            x = layers.SeparableConv2D(728, 3, padding='same')(x)
            x = layers.BatchNormalization()(x)

            x = layers.Activation('relu')(x)
            x = layers.SeparableConv2D(728, 3, padding='same')(x)
            x = layers.BatchNormalization()(x)

            x = layers.Activation('relu')(x)
            x = layers.SeparableConv2D(728, 3, padding='same')(x)
            x = layers.BatchNormalization()(x)

            x = layers.add([x, previous_block_activation])  # Add back residual
            previous_block_activation = x  # Set aside next residual

        return x
```

```
def exit_flow(x, num_classes=1000):

    previous_block_activation = x

    x = layers.Activation('relu')(x)
    x = layers.SeparableConv2D(728, 3, padding='same')(x)
    x = layers.BatchNormalization()(x)

    x = layers.Activation('relu')(x)
    x = layers.SeparableConv2D(1024, 3, padding='same')(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(3, strides=2, padding='same')(x)

    residual = layers.Conv2D(  # Project residual
      1024, 1, strides=2, padding='same')(previous_block_activation)
    x = layers.add([x, residual])  # Add back residual

    x = layers.SeparableConv2D(1536, 3, padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)

    x = layers.SeparableConv2D(2048, 3, padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)

    x = layers.GlobalAveragePooling2D()(x)
    if num_classes == 1:
        activation = 'sigmoid'
    else:
        activation = 'softmax'
    return layers.Dense(num_classes, activation=activation)(x)
```

Xây dựng model chứa 3 khối trên

```
# Xây dựng model
inputs = keras.Input(shape=(32 , 32, 3))
outputs = exit_flow(middle_flow(entry_flow(inputs)))
model = keras.Model(inputs, outputs)
```

Compile model để biên dịch model với thuật toán tối ưu là adam, hàm mất mát là sparse categorical crossentropy để model không cần thực hiện one-hot encoding với metrics là độ chính xác.

```
[ ]  # Compile model
     model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

Training model với 50 epochs (50 lần train lại tập dữ liệu liên tục).

```
# Train model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50)
```

```
Epoch 22/50
1563/1563 [==============================] - 52s 34ms/step - loss: 0.1469 - accuracy: 0.9483 - val_loss: 1.1425 - val_accuracy: 0.7535
Epoch 23/50
1563/1563 [==============================] - 52s 33ms/step - loss: 0.1941 - accuracy: 0.9328 - val_loss: 1.0703 - val_accuracy: 0.7309
Epoch 24/50
1563/1563 [==============================] - 52s 33ms/step - loss: 0.1901 - accuracy: 0.9340 - val_loss: 1.0513 - val_accuracy: 0.7606
Epoch 25/50
1563/1563 [==============================] - 53s 34ms/step - loss: 0.1129 - accuracy: 0.9613 - val_loss: 1.1947 - val_accuracy: 0.7448
Epoch 26/50
1563/1563 [==============================] - 52s 33ms/step - loss: 0.2238 - accuracy: 0.9212 - val_loss: 1.0894 - val_accuracy: 0.7623
Epoch 27/50
1563/1563 [==============================] - 52s 34ms/step - loss: 0.1369 - accuracy: 0.9523 - val_loss: 1.7160 - val_accuracy: 0.7440
Epoch 28/50
1563/1563 [==============================] - 52s 34ms/step - loss: 0.1054 - accuracy: 0.9627 - val_loss: 1.1188 - val_accuracy: 0.7551
Epoch 29/50
1563/1563 [==============================] - 52s 33ms/step - loss: 0.1101 - accuracy: 0.9611 - val_loss: 3.4192 - val_accuracy: 0.6627
Epoch 30/50
1563/1563 [==============================] - 55s 35ms/step - loss: 0.2034 - accuracy: 0.9296 - val_loss: 1.3655 - val_accuracy: 0.7484
Epoch 31/50
1563/1563 [==============================] - 53s 34ms/step - loss: 0.3654 - accuracy: 0.8746 - val_loss: 2.4261 - val_accuracy: 0.7176
Epoch 32/50
1563/1563 [==============================] - 53s 34ms/step - loss: 0.3204 - accuracy: 0.8869 - val_loss: 5.3284 - val_accuracy: 0.7411
Epoch 33/50
1563/1563 [==============================] - 54s 35ms/step - loss: 0.3314 - accuracy: 0.8851 - val_loss: 3.3173 - val_accuracy: 0.7038
Epoch 34/50
1563/1563 [==============================] - 56s 36ms/step - loss: 0.2599 - accuracy: 0.9098 - val_loss: 3.0640 - val_accuracy: 0.7542
Epoch 35/50
1563/1563 [==============================] - 55s 35ms/step - loss: 0.1219 - accuracy: 0.9574 - val_loss: 1.7441 - val_accuracy: 0.7552
```

Model cho accuracy khá cao so với các model trên với accuracy là 75,34%.

```
# Đánh giá model
model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 3s 11ms/step - loss: 2.8241 - accuracy: 0.7534
[2.8240766525268555, 0.7534000277519226]
```