به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

سیستم های هوشمند

تمرین شماره 4

نیما زمان پور

810198407

دی ماه 1401

فهرست سوالات

تمرین 1 4

Sample 1: 4

1.Predict 4

a. Forward pass 4

b. Compute Loss 5

2. Update 5

a. Back Propagation 5

b. Gradient Update 6

Sample 2: 7

1.Predict 7

a. Forward pass 7

b. Compute Loss 7

2. Update 7

a. Back Propagation 7

b. Gradient Update 8

سوال 2 11

الف) استفاده از شبکه MPL 11

1 11

2 12

3 16

4 18

5 21

ب) استفاده از شبکه های MLP+CNN: 22

1 22

2 24
3 26
4 27
سوال 3 29
الف) آشنایی با شبکه EfficientNet 29
الف) 29
ب) 30
ج) 33
د) 33
ب)پیاده سازی شبکه به کمک ایده Transfer Learning 34
ج) رفع یک مشکل خاص شبکه 34
د) آموزش شبکه با مجموعه دادگان جدید 34
پیوست 35
سوال 1) 35
سوال 2) 35
MLP) 35
CNN) 36
سوال 3) 36
2) 36
3) 36
4) 37

تمرین 1

$$810198407 \rightarrow a = 7, b = 0$$

$$\begin{cases} Z = \tanh\left(W_1^T X + B_1\right) \\ K = sigmoid(W_2^T Z + B_2) \\ P = \tanh(W_3^T K + B_3) \\ \hat{y} = ReLU(W_4^T P + B_4) \end{cases}$$

$$X_1 = \begin{bmatrix} 7 \\ 0 \\ 7 \end{bmatrix}, Y_1 = 7, X_2 = \begin{bmatrix} 0 \\ 7 \\ 0 \end{bmatrix}, Y_2 = 0$$

$$W_1 = \begin{bmatrix} 0.7 & 0 \\ 2.1 & 0 \\ 3.5 & 0 \end{bmatrix}, W_2 = \begin{bmatrix} 7.15 & 7.25 & 7.35 \\ 0.45 & 0.55 & 0.65 \end{bmatrix},$$

$$W_3 = \begin{bmatrix} 0.12 & 0.22 \\ 0.32 & 0.42 \\ 0.52 & 0.62 \end{bmatrix}, W_4 = \begin{bmatrix} 7.16 \\ 7.36 \end{bmatrix}$$

$$B_1 = \begin{bmatrix} 0.7 \\ 0 \end{bmatrix}, B_2 = \begin{bmatrix} 7.15 \\ 7.25 \\ 7.35 \end{bmatrix}, B_3 = \begin{bmatrix} 7.12 \\ 7.22 \end{bmatrix}, B_4 = 6.74$$

$$E = \frac{1}{2}(\hat{y} - y)^2, Lr = 0.1$$

Sample 1:

1.Predict

a. Forward pass

$$W_1^T X + B_1 = \begin{bmatrix} 0.7 & 2.1 & 3.5 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 7 \\ 0 \\ 7 \end{bmatrix} + \begin{bmatrix} 0.7 \\ 0 \end{bmatrix} = \begin{bmatrix} 30.1 \\ 0 \end{bmatrix}$$

$$Z = \tanh\left(\begin{bmatrix} 30.1 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$W_2^T Z + B_2 = \begin{bmatrix} 7.15 & 0.45 \\ 7.25 & 0.55 \\ 7.35 & 0.65 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 7.15 \\ 7.25 \\ 7.35 \end{bmatrix} = \begin{bmatrix} 14.3 \\ 14.5 \\ 14.7 \end{bmatrix}$$

$$K = sigmoid\left(\begin{bmatrix} 14.3 \\ 14.5 \\ 14.7 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$W_3^T K + B_3 = \begin{bmatrix} 0.12 & 0.32 & 0.52 \\ 0.22 & 0.42 & 0.62 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 7.12 \\ 7.22 \end{bmatrix} = \begin{bmatrix} 8.08 \\ 8.48 \end{bmatrix}$$

$$P = \tanh\left(\begin{bmatrix} 8.08 \\ 8.48 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$W_4^T P + B_4 = \begin{bmatrix} 7.16 \\ 7.36 \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 6.74 = 21.26$$

$$\hat{y} = ReLU(21.26) = 21.26$$

b. Compute Loss

$$E = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(21.26 - 7)^2 = 101.674$$

2. Update

a. Back Propagation

$$\frac{\partial E}{\partial W_4} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial W_4} = (\hat{y} - y) * 1 * P = \begin{bmatrix} 14.26 \\ 14.26 \end{bmatrix}$$

$$\frac{\partial E}{\partial B_4} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial B_4} = (\hat{y} - y) * 1 * 1 = 14.26$$

$$\frac{\partial E}{\partial W_{3,i,j}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial P_j} \frac{\partial P_j}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial W_{3,i}}$$

$$= (\hat{y} - y) * 1 * W_{4,j} * \text{sech}^2\left(W_{3,i,j}^T K_i + B_{3,j}\right) * K_i$$

$$\frac{\partial E}{\partial B_{3,j}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial P_j} \frac{\partial P_j}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial B_{3,j}}$$

$$= (\hat{y} - y) * 1 * W_4 * \text{sech}^2(W_3^T K + B_3) * 1$$

$$\frac{\partial E}{\partial W_{2,i,j}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial P_j} \frac{\partial P_j}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial K} \frac{\partial K}{\partial \text{sigmoid}_{\text{in}_j}} \frac{\partial \text{sigmoid}_{\text{in}_j}}{\partial W_{2,i}}$$

$$= (\hat{y} - y) * 1 * W_{4,j} * \text{sech}^2\left(W_{3,i,j}^T K_i + B_{3,j}\right) * W_3 * \left(K(1 - K)\right) * Z$$

$$\frac{\partial E}{\partial B_{2,i,j}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial P_j} \frac{\partial P_j}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial K} \frac{\partial K}{\partial \text{sigmoid}_{\text{in}_j}} \frac{\partial \text{sigmoid}_{\text{in}_j}}{\partial B_{2,i}}$$

$$= (\hat{y} - y) * 1 * W_{4,j} * \text{sech}^2\left(W_{3,i,j}^T K_i + B_{3,j}\right) * W_3 * \left(K(1 - K)\right) * Z$$

$$\frac{\partial E}{\partial W_{1,i,j}}$$

$$= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial P_j} \frac{\partial P_j}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial K} \frac{\partial K}{\partial \text{sigmoid}_{\text{in}_j}} \frac{\partial \text{sigmoid}_{\text{in}_j}}{\partial \text{Z}} \frac{\partial \text{Z}}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial W_{1,i}}$$

$$= (\hat{y} - y) * 1 * W_{4,j} * \text{sech}^2\left(W_{3,i,j}^T K_i + B_{3,j}\right) * W_3 * \left(K(1 - K)\right) * W_2$$

$$* \text{sech}^2\left(W_{2,i,j}^T K_i + B_{2,j}\right) * x$$

$$\frac{\partial E}{\partial B_{1,i,j}}$$
$$= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial P_j} \frac{\partial P_j}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial K} \frac{\partial K}{\partial \text{sigmoid}_{\text{in}_j}} \frac{\partial \text{sigmoid}_{\text{in}_j}}{\partial Z} \frac{\partial Z}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial B_{1,i}}$$
$$= (\hat{y} - y) * 1 * W_{4,j} * \text{sech}^2\left(W_{3,i,j}^T K_i + B_{3,j}\right) * W_3 * \left(K(1-K)\right) * W_2$$
$$* \text{sech}^2\left(W_{2,i,j}^T K_i + B_{2,j}\right) * 1$$

b. Gradient Update

$$W = W - \eta \nabla_w E_i, \eta = 0.1$$
$$B = B - \eta \nabla_w E_i$$
$$W_{1,1} = \begin{bmatrix} 0.7 & 0 \\ 2.1 & 0 \\ 3.5 & 0 \end{bmatrix} - 0.1 * \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.7 & 0 \\ 2.1 & 0 \\ 3.5 & 0 \end{bmatrix}$$
$$B_{1,1} = \begin{bmatrix} 0.7 \\ 0 \end{bmatrix} - 0.1 * \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0 \end{bmatrix}$$
$$W_{2,1} = \begin{bmatrix} 7.15 & 7.25 & 7.35 \\ 0.45 & 0.55 & 0.65 \end{bmatrix} - 0.1 * \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 7.15 & 7.25 & 7.35 \\ 0.45 & 0.55 & 0.65 \end{bmatrix}$$
$$B_{2,1} = \begin{bmatrix} 7.15 \\ 7.25 \\ 7.35 \end{bmatrix} - 0.1 * \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 7.15 \\ 7.25 \\ 7.35 \end{bmatrix}$$
$$W_{3,1} = \begin{bmatrix} 0.12 & 0.22 \\ 0.32 & 0.42 \\ 0.52 & 0.62 \end{bmatrix} - 0.1 * \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.12 & 0.22 \\ 0.32 & 0.42 \\ 0.52 & 0.62 \end{bmatrix}$$
$$B_{3,1} = \begin{bmatrix} 7.12 \\ 7.22 \end{bmatrix} - 0.1 * \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 7.12 \\ 7.22 \end{bmatrix}$$
$$W_{4,1} = \begin{bmatrix} 7.16 \\ 7.36 \end{bmatrix} - 0.1 * \begin{bmatrix} 14.26 \\ 14.26 \end{bmatrix} = \begin{bmatrix} 5.734 \\ 5.934 \end{bmatrix}$$
$$B_{4,1} = 6.74 - 0.1 * 14.26 = 5.314$$

هر 4 مرحله در روش Gradient descent انجام شد. وزن ها آپدیت شدند. به دلیل بزرگ بودن ورودی های توابع فعال ساز tanh, sigmoid این توابع اشباع شدند. و وزن های لایه های ما قبل آخر آپدیت نتوانستند بشوند.

Sample 2:

1.Predict

a. Forward pass

$$W_1^T X + B_1 = \begin{bmatrix} 0.7 & 2.1 & 3.5 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 7 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.7 \\ 0 \end{bmatrix} = \begin{bmatrix} 15.4 \\ 0 \end{bmatrix}$$

$$Z = \tanh\left(\begin{bmatrix} 15.4 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$W_2^T Z + B_2 = \begin{bmatrix} 7.15 & 0.45 \\ 7.25 & 0.55 \\ 7.35 & 0.65 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 7.15 \\ 7.25 \\ 7.35 \end{bmatrix} = \begin{bmatrix} 14.3 \\ 14.5 \\ 14.7 \end{bmatrix}$$

$$K = sigmoid\left(\begin{bmatrix} 14.3 \\ 14.5 \\ 14.7 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$W_3^T K + B_3 = \begin{bmatrix} 0.12 & 0.32 & 0.52 \\ 0.22 & 0.42 & 0.62 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 7.12 \\ 7.22 \end{bmatrix} = \begin{bmatrix} 8.08 \\ 8.48 \end{bmatrix}$$

$$P = \tanh\left(\begin{bmatrix} 8.08 \\ 8.48 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$W_4^T P + B_4 = \begin{bmatrix} 5.734 \\ 5.934 \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 5.314 = 16.98$$

$$\hat{y} = ReLU(16.98) = 16.98$$

b. Compute Loss

$$E = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(16.98 - 0)^2 = 144.19$$

2. Update

a. Back Propagation

$$\frac{\partial E}{\partial W_4} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial W_4} = (\hat{y} - y) * 1 * P = \begin{bmatrix} 16.98 \\ 16.98 \end{bmatrix}$$

$$\frac{\partial E}{\partial B_4} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial B_4} = (\hat{y} - y) * 1 * 1 = 16.98$$

$$\frac{\partial E}{\partial W_{3,i,j}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial P_j} \frac{\partial P_j}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial W_{3,i}}$$

$$= (\hat{y} - y) * 1 * W_{4,j} * \operatorname{sech}^2\left(W_{3,i,j}^T K_i + B_{3,j}\right) * K_i$$

$$\frac{\partial E}{\partial B_{3,j}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial P_j} \frac{\partial P_j}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial B_{3,j}}$$

$$= (\hat{y} - y) * 1 * W_4 * \operatorname{sech}^2(W_3^T K + B_3) * 1$$

$$\frac{\partial E}{\partial W_{2,i,j}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial P_j} \frac{\partial P_j}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial K} \frac{\partial K}{\partial \text{sigmoid}_{\text{in}_j}} \frac{\partial \text{sigmoid}_{\text{in}_j}}{\partial W_{2,i}}$$

$$= (\hat{y} - y) * 1 * W_{4,j} * \operatorname{sech}^2\left(W_{3,i,j}^T K_i + B_{3,j}\right) * W_3 * \left(K(1 - K)\right) * Z$$

$$\frac{\partial E}{\partial B_{2,i,j}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial P_j} \frac{\partial P_j}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial K} \frac{\partial K}{\partial \text{sigmoid}_{\text{in}_j}} \frac{\partial \text{sigmoid}_{\text{in}_j}}{\partial B_{2,i}}$$
$$= (\hat{y} - y) * 1 * W_{4,j} * \text{sech}^2\left(W_{3,i,j}^T K_i + B_{3,j}\right) * W_3 * \left(K(1-K)\right) * Z$$

$$\frac{\partial E}{\partial W_{1,i,j}}$$
$$= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial P_j} \frac{\partial P_j}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial K} \frac{\partial K}{\partial \text{sigmoid}_{\text{in}_j}} \frac{\partial \text{sigmoid}_{\text{in}_j}}{\partial Z} \frac{\partial Z}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial W_{1,i}}$$
$$= (\hat{y} - y) * 1 * W_{4,j} * \text{sech}^2\left(W_{3,i,j}^T K_i + B_{3,j}\right) * W_3 * \left(K(1-K)\right) * W_2$$
$$* \text{sech}^2\left(W_{2,i,j}^T K_i + B_{2,j}\right) * x$$

$$\frac{\partial E}{\partial B_{1,i,j}}$$
$$= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial ReLU_{in}} \frac{\partial ReLU_{in}}{\partial P_j} \frac{\partial P_j}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial K} \frac{\partial K}{\partial \text{sigmoid}_{\text{in}_j}} \frac{\partial \text{sigmoid}_{\text{in}_j}}{\partial Z} \frac{\partial Z}{\partial \tanh_{\text{in}_j}} \frac{\partial \tanh_{\text{in}_j}}{\partial B_{1,i}}$$
$$= (\hat{y} - y) * 1 * W_{4,j} * \text{sech}^2\left(W_{3,i,j}^T K_i + B_{3,j}\right) * W_3 * \left(K(1-K)\right) * W_2$$
$$* \text{sech}^2\left(W_{2,i,j}^T K_i + B_{2,j}\right) * 1$$

b. Gradient Update

$$W = W - \eta \nabla_w E_i, \eta = 0.1$$

$$B = B - \eta \nabla_w E_i$$

$$W_{1,2} = \begin{bmatrix} 0.7 & 0 \\ 2.1 & 0 \\ 3.5 & 0 \end{bmatrix} - 0.1 * \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.7 & 0 \\ 2.1 & 0 \\ 3.5 & 0 \end{bmatrix}$$

$$B_{1,2} = \begin{bmatrix} 0.7 \\ 0 \end{bmatrix} - 0.1 * \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0 \end{bmatrix}$$

$$W_{2,2} = \begin{bmatrix} 7.15 & 7.25 & 7.35 \\ 0.45 & 0.55 & 0.65 \end{bmatrix} - 0.1 * \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 7.15 & 7.25 & 7.35 \\ 0.45 & 0.55 & 0.65 \end{bmatrix}$$

$$B_{2,2} = \begin{bmatrix} 7.15 \\ 7.25 \\ 7.35 \end{bmatrix} - 0.1 * \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 7.15 \\ 7.25 \\ 7.35 \end{bmatrix}$$

$$W_{3,2} = \begin{bmatrix} 0.12 & 0.22 \\ 0.32 & 0.42 \\ 0.52 & 0.62 \end{bmatrix} - 0.1 * \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.12 & 0.22 \\ 0.32 & 0.42 \\ 0.52 & 0.62 \end{bmatrix}$$

$$B_{3,2} = \begin{bmatrix} 7.12 \\ 7.22 \end{bmatrix} - 0.1 * \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 7.12 \\ 7.22 \end{bmatrix}$$

$$W_{4,1} = \begin{bmatrix} 5.734 \\ 5.934 \end{bmatrix} - 0.1 * \begin{bmatrix} 16.98 \\ 16.98 \end{bmatrix} = \begin{bmatrix} 4.036 \\ 4.236 \end{bmatrix}$$

$$B_{4,1} = 5.314 - 0.1 * 16.98 = 3.616$$

با پیاده‌سازی شبکه به کمک کتابخانه pytorch وزن ها و بایاس ها همان بدست آمد.

```
Weights biases after first update:
W1:
[[ 0.7 -0. ]
 [ 2.1  0. ]
 [ 3.5 -0. ]]
B1:
[[ 0.7]
 [-0. ]]
W2:
[[7.15 7.25 7.35]
 [0.45 0.55 0.65]]
B2:
[[7.15]
 [7.25]
 [7.35]]
W3:
[[0.12 0.22]
 [0.32 0.42]
 [0.52 0.62]]
B3:
[[7.12]
 [7.22]]
W4:
[[5.734]
 [5.934]]
B4:
[5.314]
```

Figure 1 وزن ها و بایاس های بعد از اولین آپدیت

```
Weights biases after second update:
W1:
[[ 0.7 -0. ]
 [ 2.1 -0. ]
 [ 3.5 -0. ]]
B1:
[[ 0.7]
 [-0. ]]
W2:
[[7.15 7.25 7.35]
 [0.45 0.55 0.65]]
B2:
[[7.15]
 [7.25]
 [7.35]]
W3:
[[0.12 0.22]
 [0.32 0.42]
 [0.52 0.62]]
B3:
[[7.12]
 [7.22]]
W4:
[[4.036]
 [4.236]]
B4:
[3.616]
```

Figure 2 وزن ها و بایاس ها بعد از دومین آپدیت

سوال 2

بعد از دانلود دیتاست و تقسیم آن به دسته های آموزش و تست و ارزیابی، 10 تصویر از دیتاست به را به عنوان نمونه نمایش می‌دهیم:

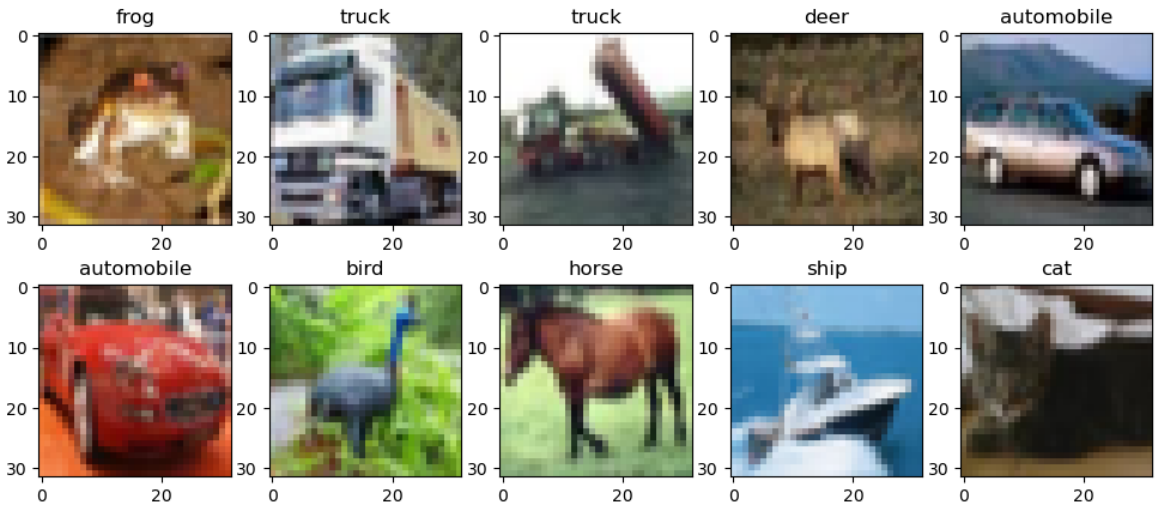

Figure 3 10 نمونه تصویر از دیتاست

همانطور که می‌بینیم ورودی عکس ها 32*32 3 کاناله رنگی است.

برای پیش پردازش داده ها ابتدا عکس ها را بصورت بردار کرده تا قابل ورودی دادن به مدل باشد. سپس لیبل ها را بصورت one-hot انکود می‌کنیم.

الف) استفاده از شبکه MPL

.1

در مدل با بالا رفتن اندازه دسته ها سرعت مدل بیشتر می‌شود. چون Iteration های کمتری در هر epoch انجام می‌شود. ولی از طرفی دقت مدل پایین می‌آید چون گرادیان های کمتری می‌گیرد. دقت ها برای batch size = 256, 64, 32 به ترتیب برابر ٪33.65 و ٪30.58 و ٪22.41 است. همچنین زمان اجرا به ترتیب برابر 2:11 و 0:57 و 0:21 دقیقه شد.

```
model = Sequential()
model.add(Input(shape = (32*32*3)))
model.add(Dense(512, activation='tanh'))
model.add(Dense(512, activation='relu'))
model.add(Dense(10  , activation='softmax'))
model.compile(loss = 'mse', optimizer = tf.keras.optimizers.SGD(learning_rate=0.001), metrics = ['accuracy'])

batch_size = 32
epochs = 20
history = model.fit(x_train_mlp, y_train_cat,
          batch_size = batch_size,
          epochs = epochs,
          verbose = 1,
          validation_data = (x_test_mlp, y_test_cat))
```

Figure 4 مدل اولیه شبکه

.2

برای تغییر اول توابع فعال ساز لایه‌ها را به relu ,relu, softmax تغییر می‌دهیم. مدل به دقت ٪19.60 و خطای 0.1602 می‌رسد.

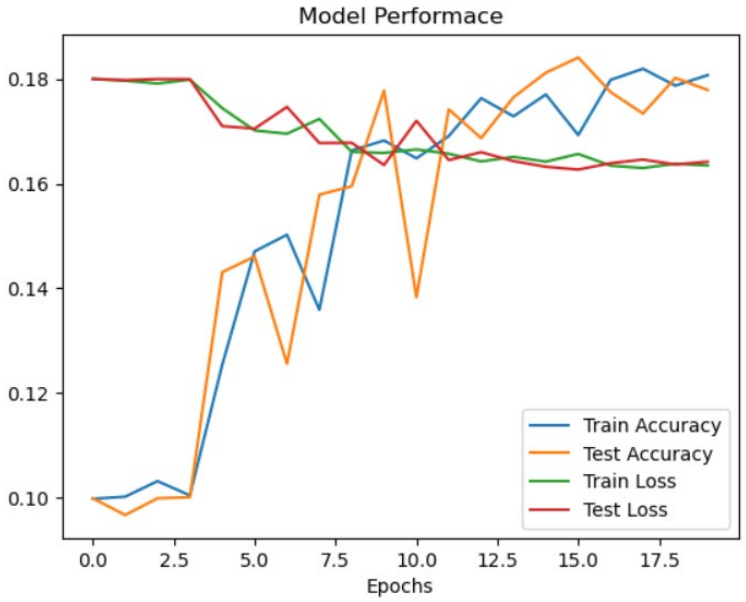

Figure 5 عملکرد مدل برای توابع فعال ساز حالت اول

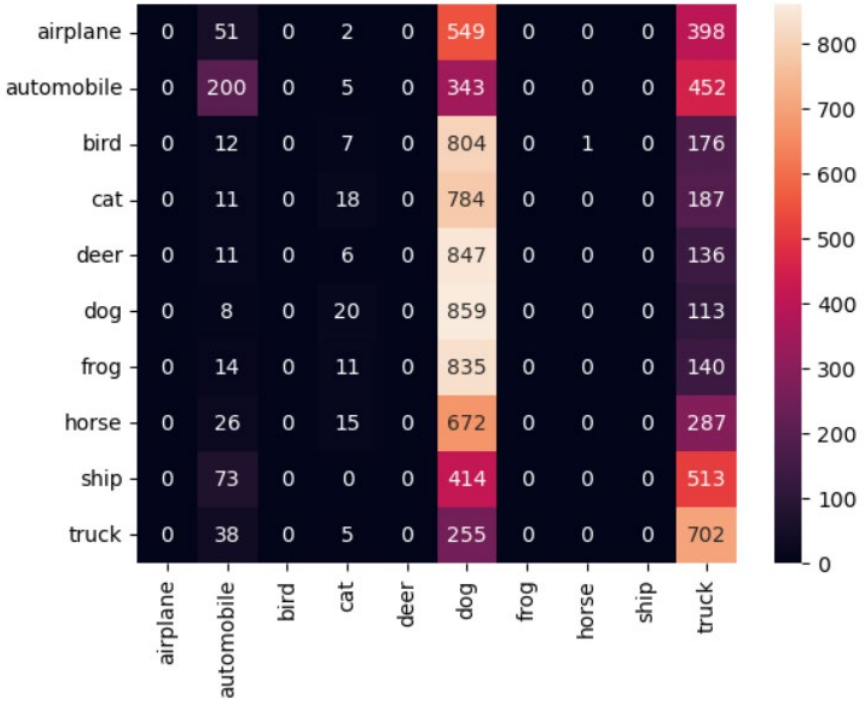

Figure 6 ماتریس آشفتگی مدل برای توابع فعال ساز حالت اول

برای تغییر دوم توابع فعال ساز لایه‌ها را به tanh, sigmoid, softmax تغییر می‌دهیم. مدل به دقت 26.38٪ و خطای 0.0860 می‌رسد.

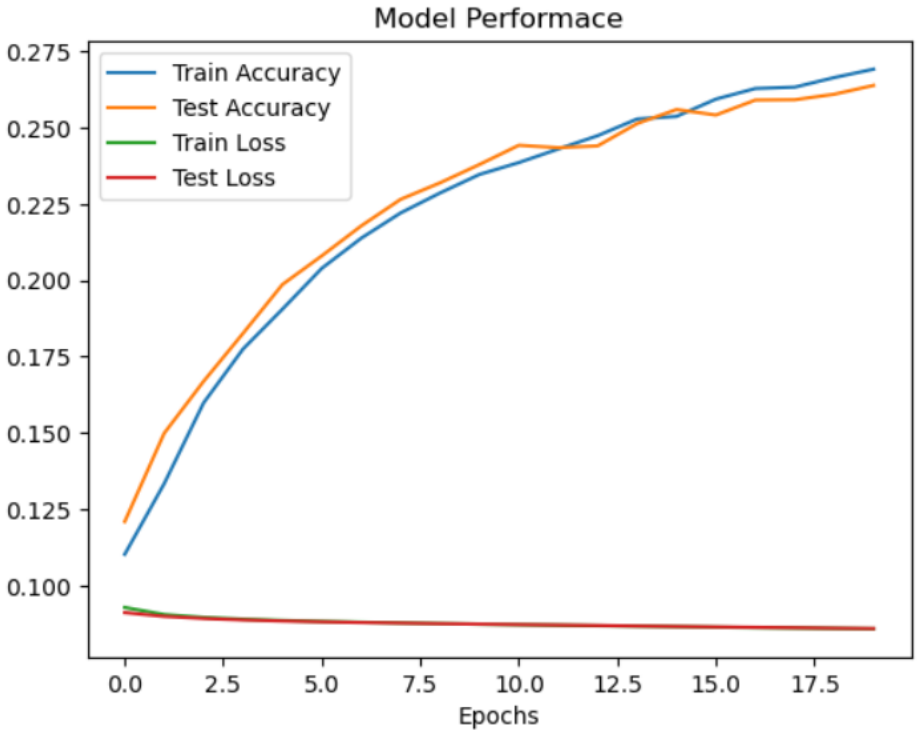

Figure 7 عملکرد مدل برای توابع فعال ساز حالت دوم

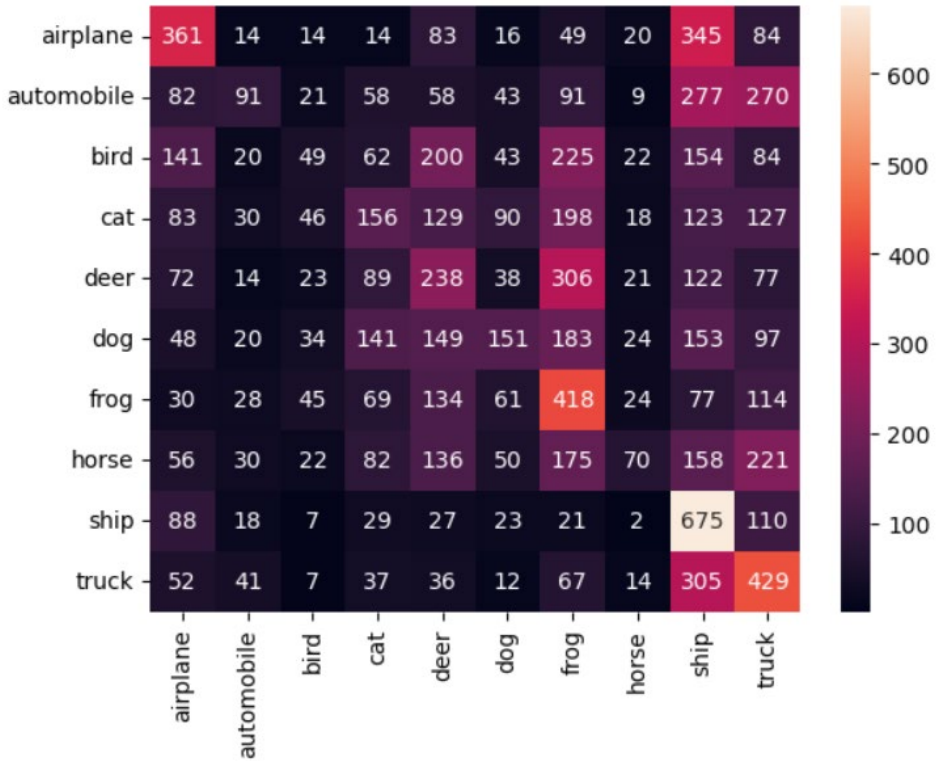

Figure 8 ماتریس آشفتگی مدل برای توابع فعال ساز حالت دوم

برای تغییر سوم توابع فعال ساز لایه‌ها را به sigmoid, relu, softmax تغییر می‌دهیم. مدل به دقت 29.88٪ و خطای 0.0826 می‌رسد.

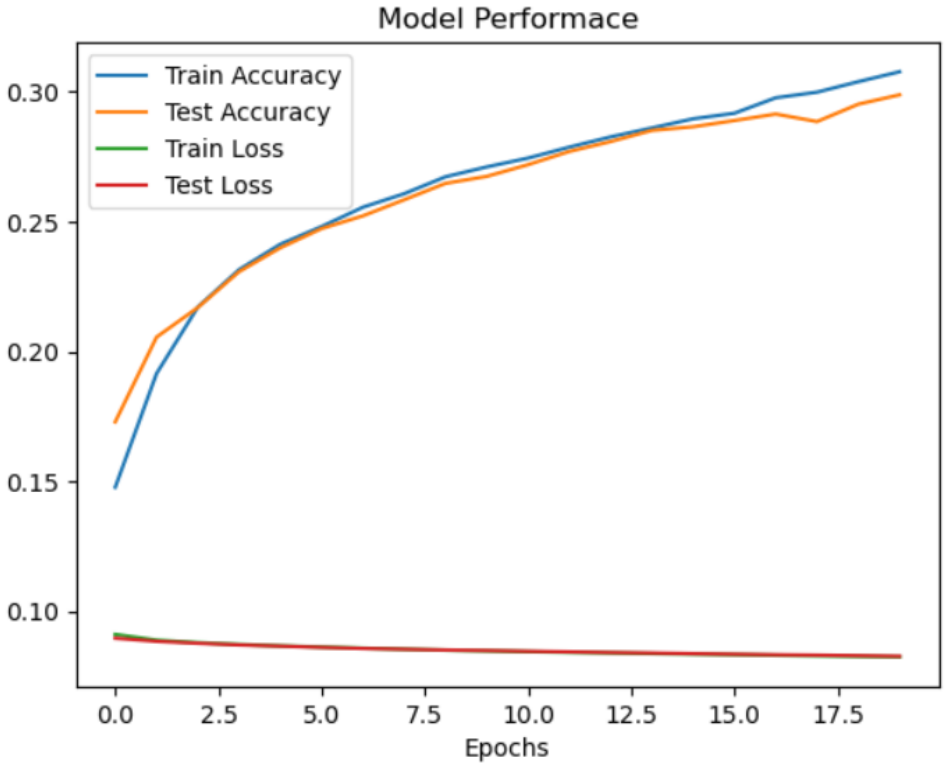

Figure 9 عملکرد مدل برای توابع فعال ساز حالت سوم

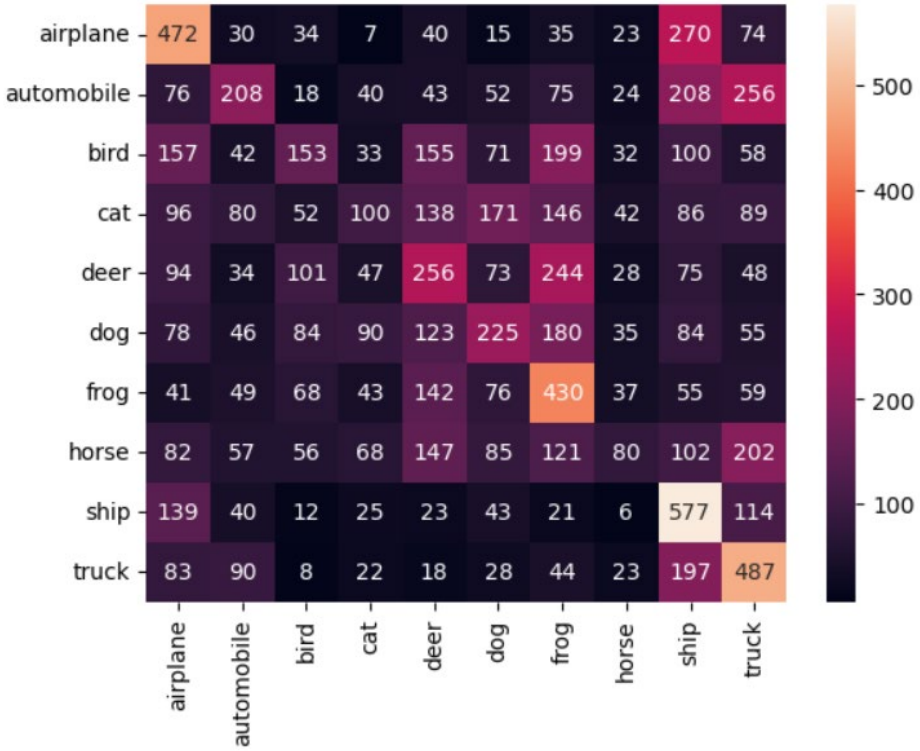

Figure 10 ماتریس آشفتگی مدل برای توابع فعال ساز حالت سوم

در بررسی مزایا و معایب توابع فعالساز ابتدا از تابع $sigmoid = \frac{1}{1+e^{-x}}$ شروع می‌کنیم. این تابع به راحتی در مقادیر بالا اشباع شده و باعث می‌شود. گرادیان های ورودی انتقال پیدا نکنند.(vanishing gradient problem) مشکل دوم این است که چون sigmoid در صفر zero centered نیست. و مقایر همیشه مثبت دارد. پس در گرادیان گرفتن یا همیشه مثبت است یا همیشه منفی. و در نزول کردن به نقطه بهینه زیگزاگ رفتار می‌کند. که همگرایی را کند می‌کند. همچنین محاسبه exp() از نظر محاسباتی طولانی است.

تابع بعدی tanh است. که عیب zero centered نبودن را ندارد. و سریعتر است. ولی همچنان گرادیان ها را اشباع می‌کند.

تابع بعدی ReLu است. که رکتیفایر خطی است. این تابع بهترین تابع فعال ساز است. در قسمت مثبت ها هیچ وقت اشباع نمی‌شود. پیاده سازی آن راحت و سریع است. اما عیب آن این است که در هنگام حساب کردن گرادیان اگر ورودی آن منفی باشد. مشتق صفر شده و آن نورون آپدیت نمی‌شود.(dead neuron)

.3

تابع خطای شبکه را از mse به categorical crossentropy و hinge تغییر دادیم.

$$CCE\ Loss = -\frac{1}{n_{class}} \sum_{i=1}^{n_{class}} y_i \log \hat{y_i} + (1 - y_i) \log(1 - \hat{y_i})$$

برای تابع خطای CCE دقت به ٪35.61 و خطا به 1.8242 رسید.

تابع CCE به دلیل آنکه عملکرد بهتری در محاسبه احتمال درست بودن هر کلاس در بازه 0 تا 1 دارد. قسمت$(1 - y_i) \log(1 - \hat{y_i})$ $y_i \log \hat{y_i} +$ احتمالات غلط را بیشتر از ترم $(y_i - \hat{y_i})^2$ در MSE جریمه می‌کند. چون مقدار log( ) با اختلاف های کم، بیشتر زیاد می‌شود. به همین دلیل loss بیشتری دارد. و بهتر عمل می‌کند.

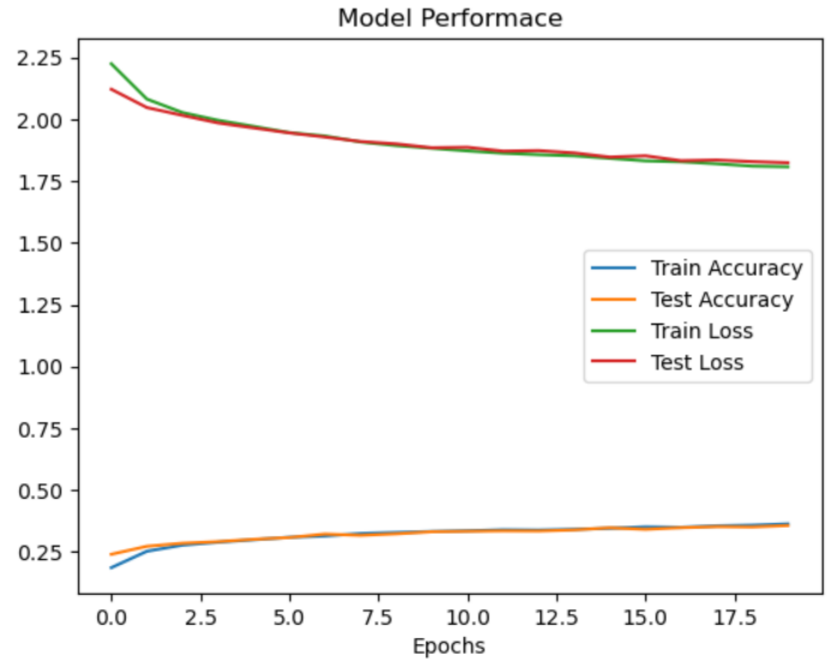

Figure 11 عملکرد مدل با تابع خطای **CCE**

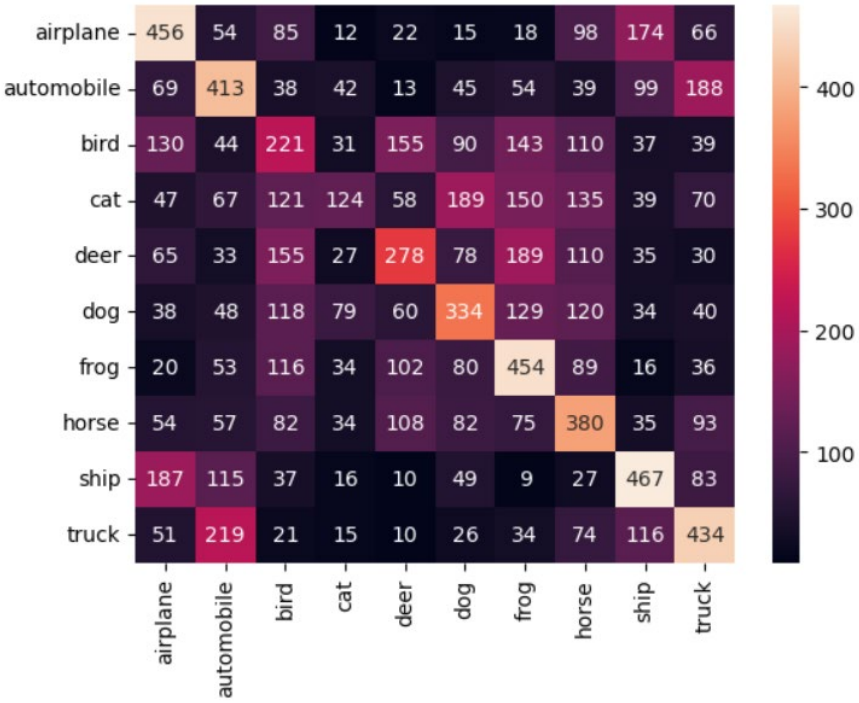

Figure 12 ماتریس آشفتگی مدل با تابع خطای **CCE**

$$hinge\ Loss = \max(0, 1 - y\hat{y}_i)$$

برای تابع خطای hinge دقت به ٪ 27.09 و خطا به 1.0533 رسید.

از آنجایی که مقدار های $y, \hat{y}_i$ مقدار بین صفر و یک دارند پس همیشه این $1 - y\hat{y}_i \geq 0$ و خروجی تابع مقداری کمتر و نزدیک 1 است. پس با جمع همه کلاس ها و سپس میانگین گرفتن آن ها loss تابع نیز حدود 1 می‌شود.(پیاده سازی دقیق این تابع در کتابخانه با فرمول اصلی آن کاملا برابر نیست).

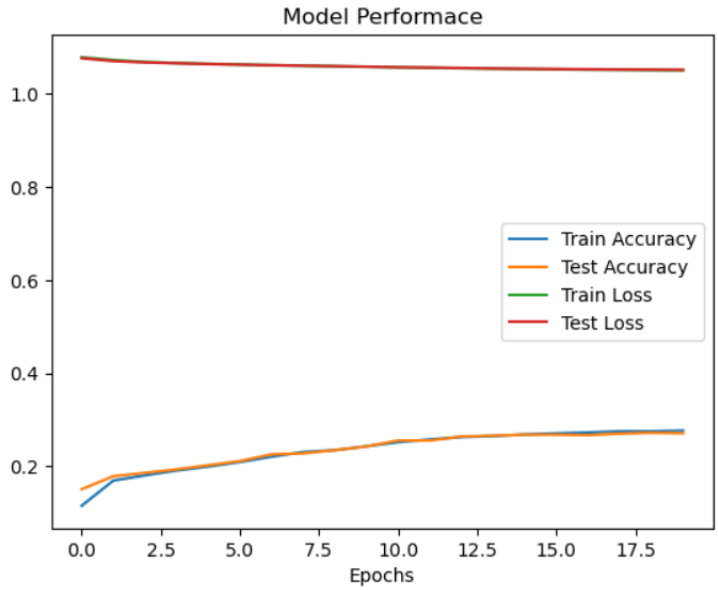

Figure 13 عملکرد مدل با تابع خطای **hinge**

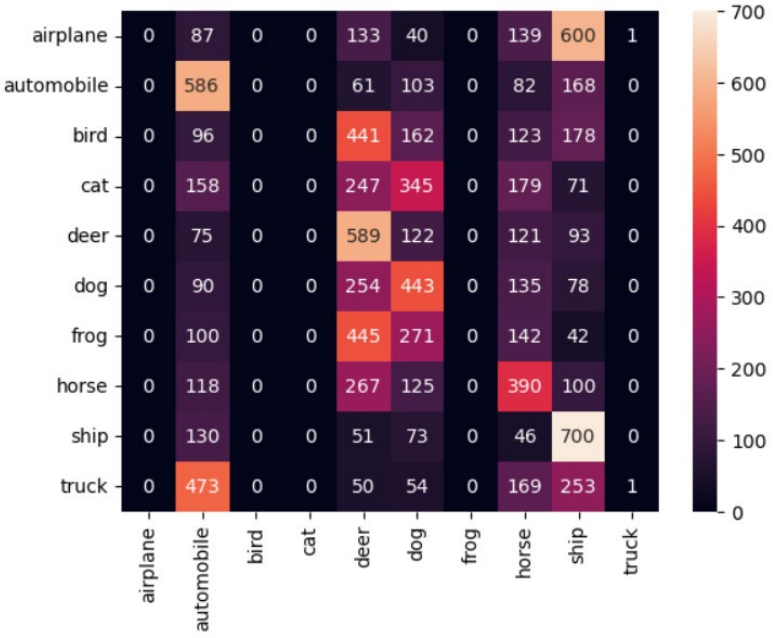

Figure 14 ماتریس آشفتگی مدل با تابع خطای **hinge**

.4

ابتدا بهینه ساز Adam را بررسی می‌کنیم. این بهینه ساز به دقت 39.47٪ و خطای 1.7933 رسید. که تا اینجا از همه شبکه ها بهتر بود.

برای عملکرد بهتر بهینه ساز adam و adadelta در ادامه، توابع فعال سازی که برای SGD بهتر عمل می‌کرد تغییر داده شد تا عملکرد adam بهینه شود. و نتیجه آن، دقت 2 برابری برای مدل هنگام تعویض از tanh, relu, softmax به relu, relu, softmax بود.

بهینه ساز adam همان روش SGD است با این فرق که از مومنتوم درجه یک و درجه 2 بصورت تخمینی پویا برای آپدیت استفاده می‌کند.

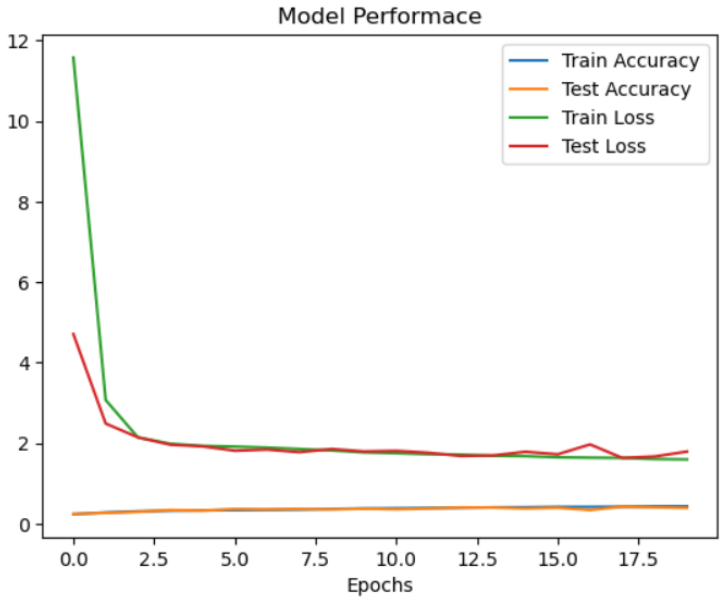

Figure 15 عملکرد مدل با بهینه ساز **adam**

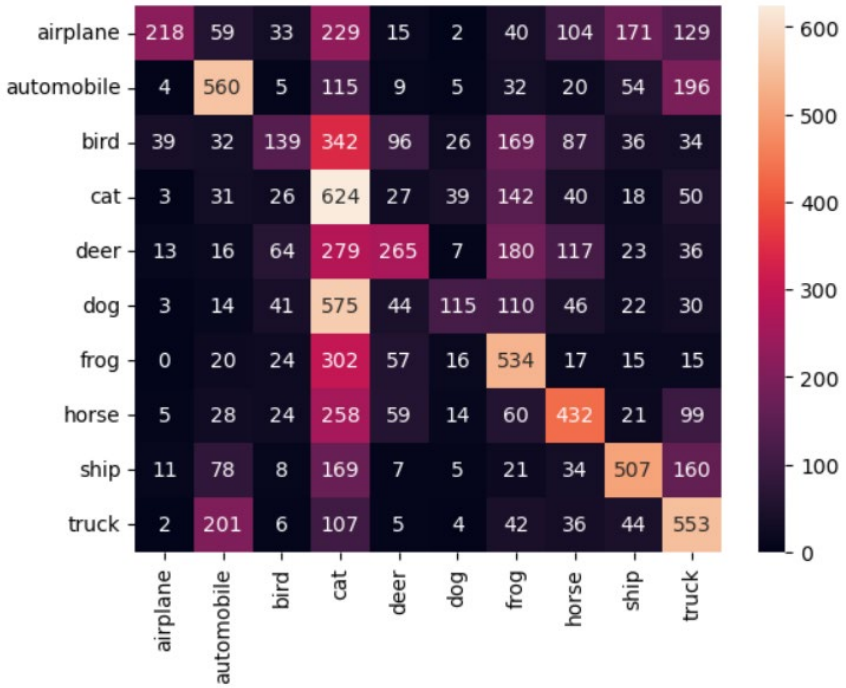

Figure 16 ماتریس آشتفگی مدل با بهینه ساز **adam**

بهینه ساز دوم adadelta است. که با اجرا به دقت ٪28.76 و خطای 10.8574 رسید.

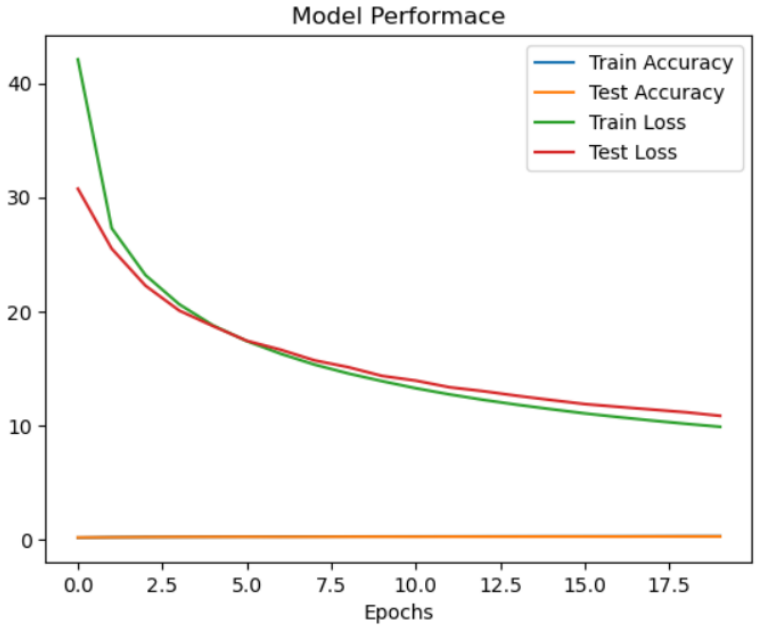

Figure 17 عملکرد مدل با بهینه ساز **adadelta**

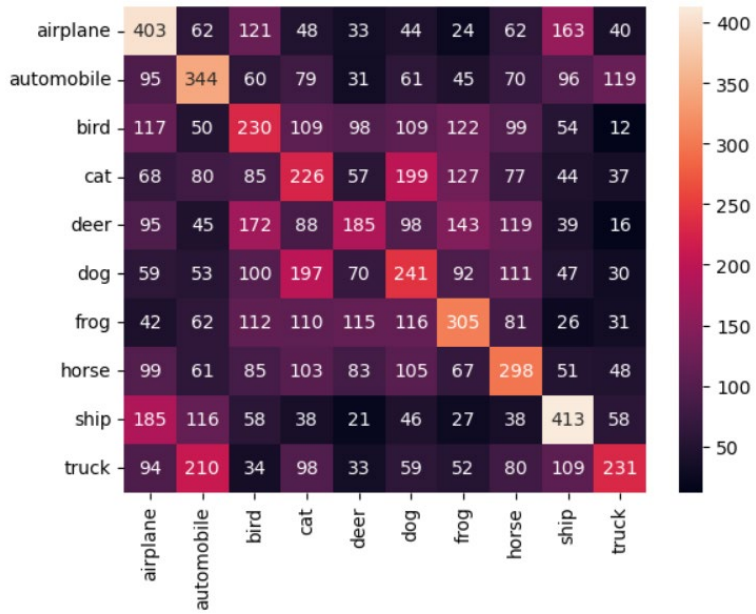

Figure 18 ماتریس آشفتگی مدل با بهینه ساز **adadelta**

بهینه ساز adadelta همان روش SGD است با این فرق که برای هر بعد یک نرخ یادگیری جداگانه دارد. و در مقایسه با بقیه روش ها نرخ همگرایی آن در گذر زمان به صفر میل نمی‌کند. و نیاز به یک نرخ یادگیری برای همه ابعاد را از بین می‌برد.

.5

بر اساس ارزیابی های انجام شده، مدل با پارامتر های زیر بهترین عملکرد را دارد:

```
best_model = Sequential()
best_model.add(Input(shape = (32*32*3)))
best_model.add(Dense(512, activation='relu'))
best_model.add(Dense(512, activation='relu'))
best_model.add(Dense(10  , activation='softmax'))
best_model.compile(loss = 'categorical_crossentropy', optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001), metrics = ['accuracy'])

batch_size = 32
epochs = 20
best_history = best_model.fit(x_train_mlp, y_train_cat,
          batch_size = batch_size,
          epochs = epochs,
          verbose = 1,
          validation_data = (x_test_mlp, y_test_cat))
```

1m 56.7s

Figure 19 بهترین مدل با معماری **MLP**

	precision	recall	f1-score	support
0	0.73	0.22	0.34	1000
1	0.54	0.56	0.55	1000
2	0.38	0.14	0.20	1000
3	0.21	0.62	0.31	1000
4	0.45	0.27	0.33	1000
5	0.49	0.12	0.19	1000
6	0.40	0.53	0.46	1000
7	0.46	0.43	0.45	1000
8	0.56	0.51	0.53	1000
9	0.42	0.55	0.48	1000
accuracy			0.39	10000
macro avg	0.46	0.39	0.38	10000
weighted avg	0.46	0.39	0.38	10000

Figure 20 سایر معیار های ارزیابی: **F1-score, recall, precision**

در نهایت با ارزیابی مدل روی داده های validation به دقت ٪40.46 می‌رسیم.

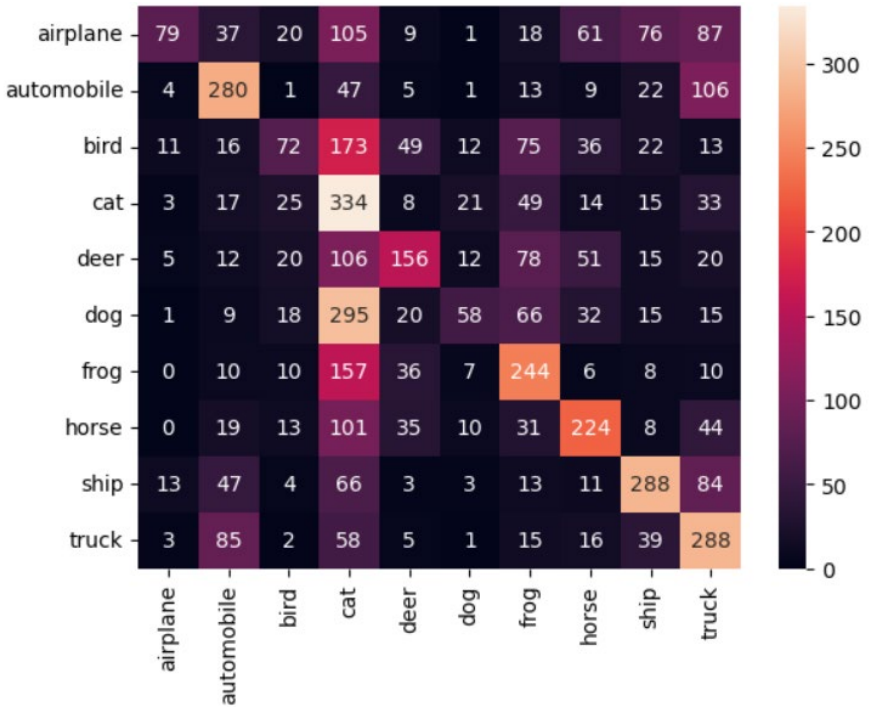

Figure 21 ماتریس آشفتگی بر روی داده‌های **validation**

ب) استفاده از شبکه های MLP+CNN:

.1

ابتدا داده ها را standard scale کرده. سپس شبکه را با اضافه کردن 3 لایه conv2d تکمیل کرده و با پیاده سازی آن به دقت 62.89٪ و خطای 2.5255 رسید.

شبکه از نظر دقت بهبود چشمگیری نسبت به MLP داشته است. اما خطای آن بیشتر شده است. که به دلیل overfit شدن شبکه است.(به دلیل داشتن پارامتر های بسیار بیشتر)

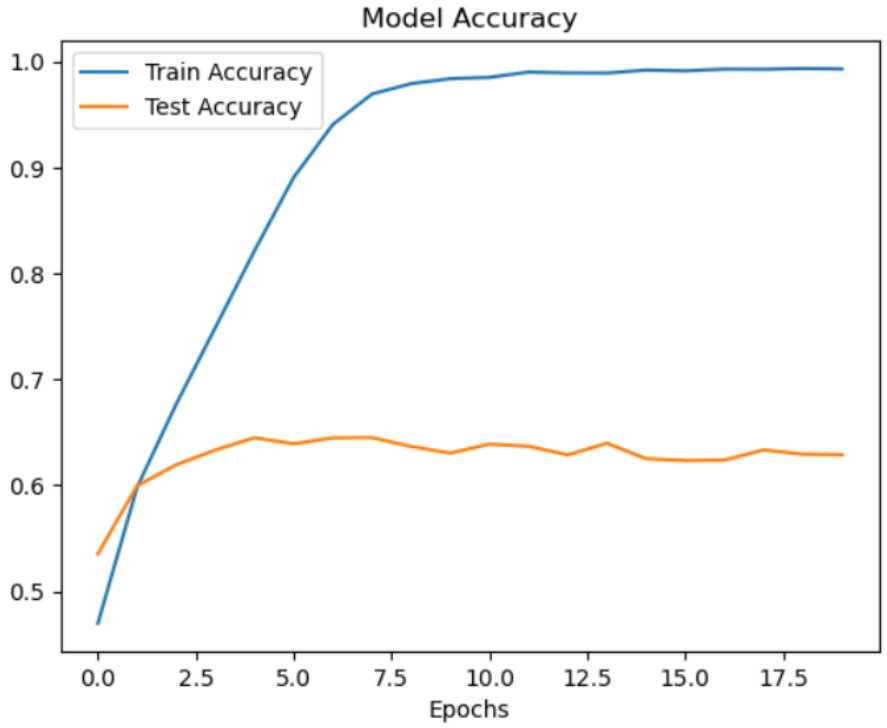

Figure 22 نمودار دقت شبکه **CNN**

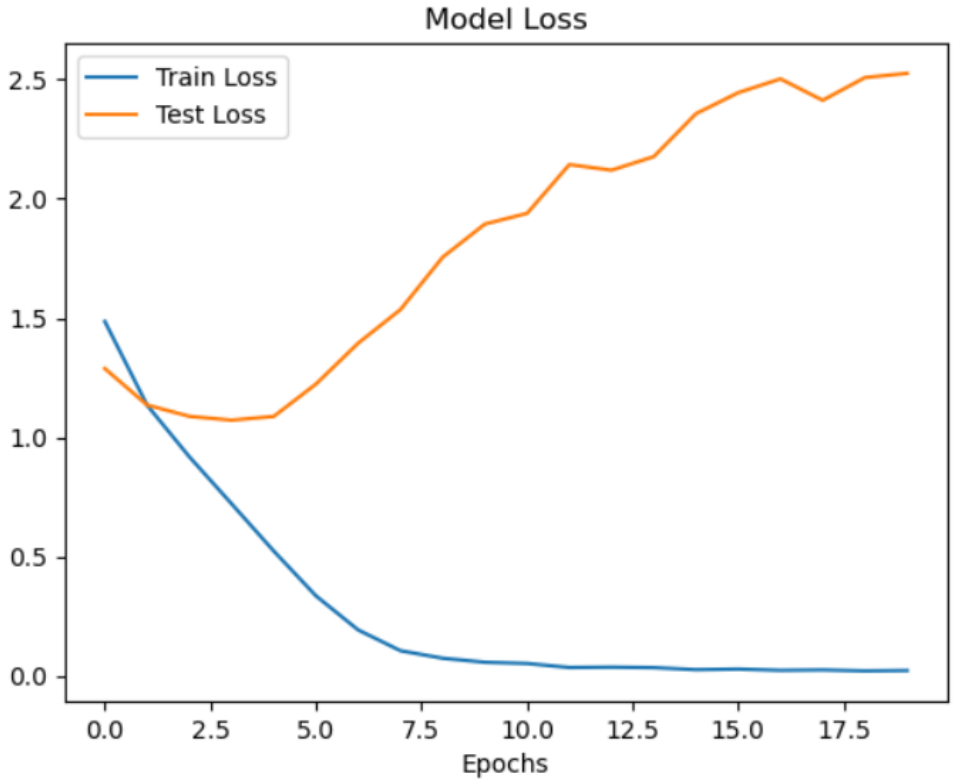

Figure 23 نمودار خطای شبکه **CNN**

.2

در یک شبکه هنگامی که داده ها نرمال هستند. و توزیع آن ها در لایه های مختلف مشابه است. سریع است. با شروع به آموزش شبکه، وزن های لایه ها و توزیع ورودی آن ها و به تبع خروجی‌شان تغییر می‌کند. که به آن internal covariate shift می‌گویند. لایه batch normalization ورودی هر لایه برای هر mini-batch را استاندارد می‌کند.(میانگین 0 و واریانس 1) این کار باعث می‌شود که شبکه سریعتر آموزش داده شود. و بتوان با نرخ های یادگیری به مراتب بزرگتری در epoch کمتری آموزش داد.

لایه pooling یک فیلتر 2 بعدی برای کاهش اندازه هر بعد و خلاصه کردن ویژگی های نهفته درون ماتریس(معمولا عکس) است. با کاهش اندازه ماتریس، پارامتر های لازم برای آموزش شبکه کاهش می‌یابد. این لایه معمولا بعد از هر لایه convolutional قرار می‌گیرد.

2 نوع معمول pooling داریم. Max pooling که در هر بار اسکن ماتریس، بزرگترین درایه ای که در scope فیلتر آن قرار دارد را برمی‌گرداند. نوع دیگر average pooling است. که میانگین درایه های دیده شده در هر scope را برمی‌گرداند. این لایه 2 آرگومان دارد که یکی سایز فیلتر و دیگری stride که مقدار جابجا شدن فیلتر در هنگام پیمایش روی ماتریس است.

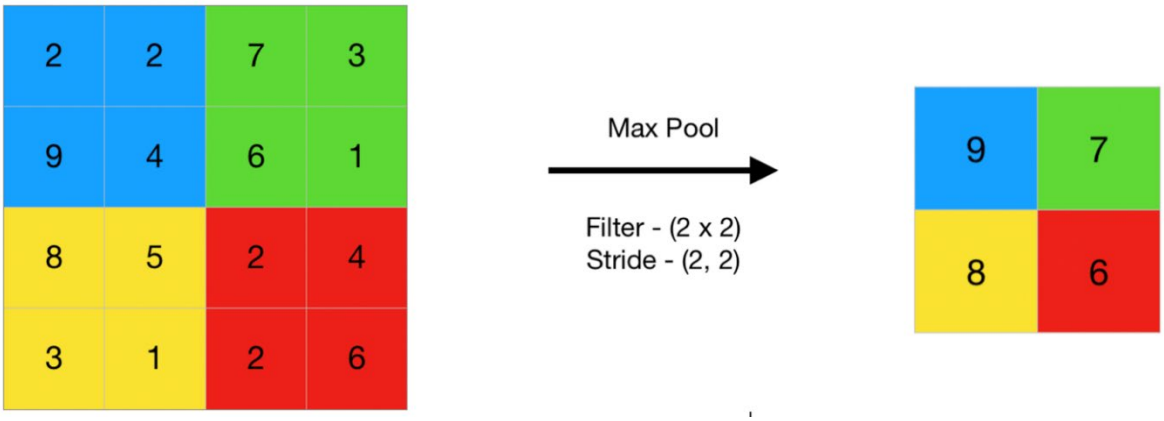

Figure 24 نحوه اسکن کردن در روش **max pooling** منبع

با اضافه کردن این لایه ها و تغییر نرخ یادگیری به 0.001 (به دلیل وجود لایه batch normalization میتوان با اطمینان نرخ یادگیری را تغییر داد و سریعتر همگرا شد) دقت مدل به ٪72.95 رسید. و خطای آن 1.0247 شد.

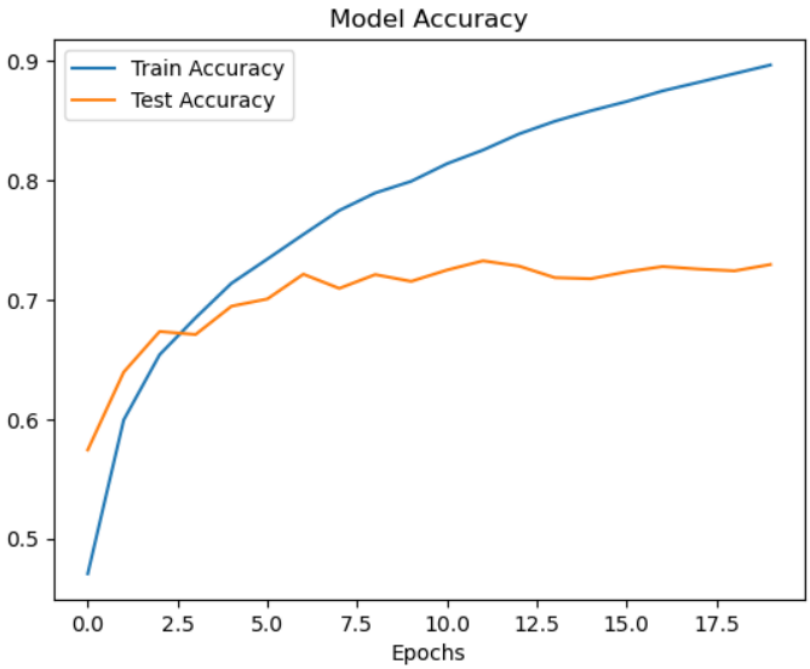

Figure 25 دقت مدل با اضافه کردن لایه های **pooling, BN**

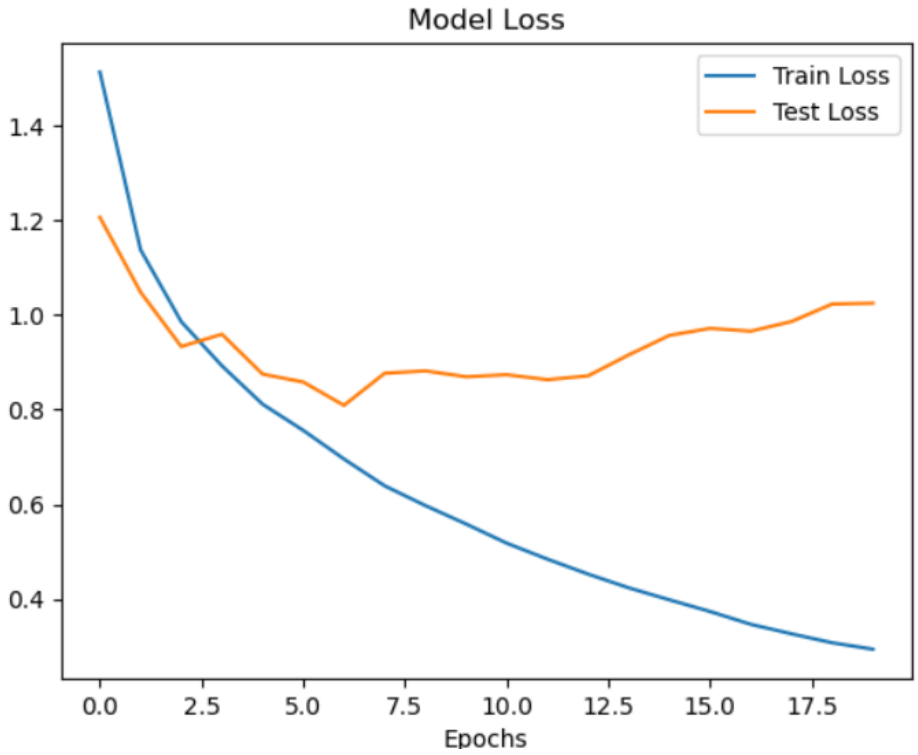

Figure 26 خطای مدل با اضافه کردن لایه های **pooling, BN**

دقت مدل بیشتر و خطای آن کمتر شد. مدل همچنان overfit است. اما نسبت به سوال 1 فاصله دقت داده های آموزش و آزمون کمتر شد. لایه BN باعث شد که بتوان نرخ یادگیری را بالا برد و به کمک آن در 20 epoch مدل آموزش بیشتری دید.

.3

با اضافه کردن dropout زیر هر لایه pooling دقت مدل به ٪72.53 و خطای آن 0.7845 شد.

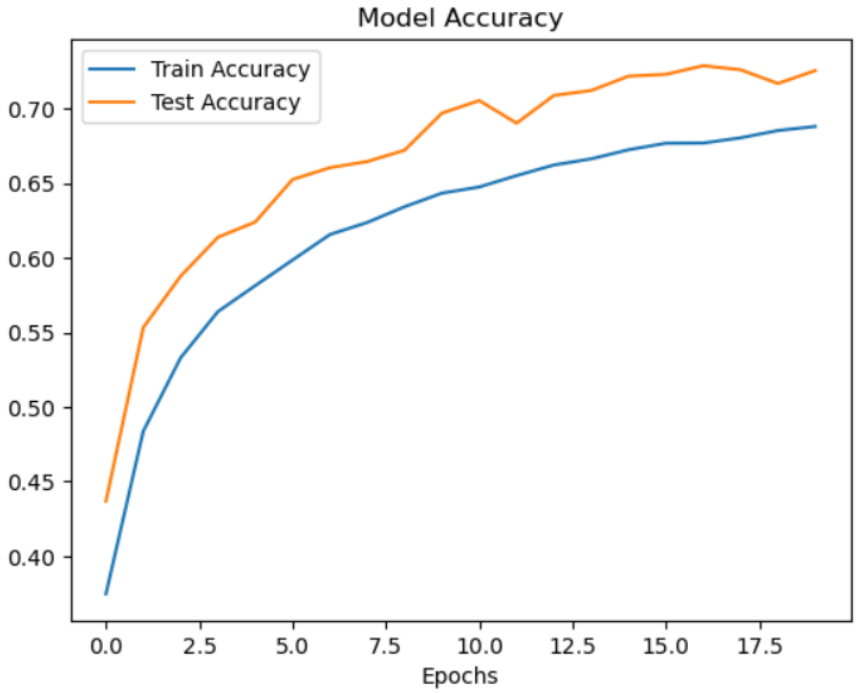

Figure 27 نمودار دقت مدل با اضافه کردن لایه **dropout**

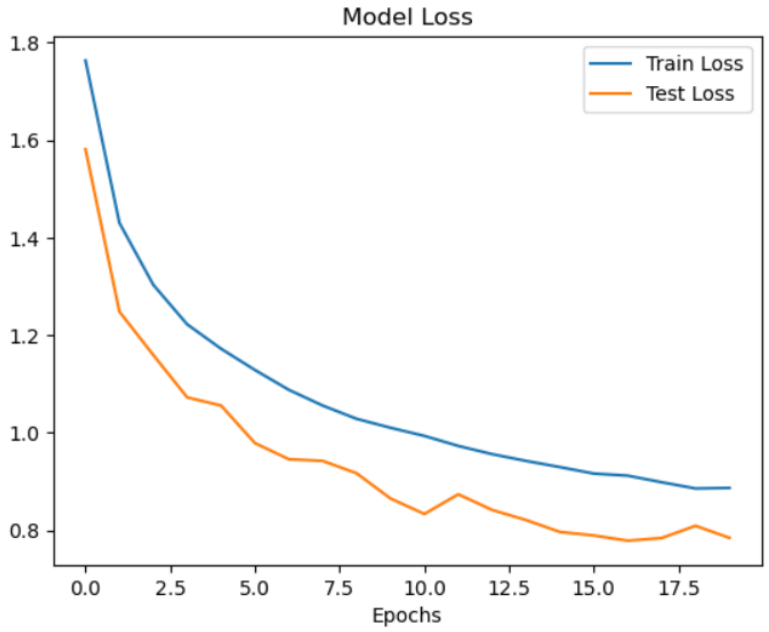

Figure 28 خطای مدل با اضافه کردن لایه **dropout**

همانطور که در شکل پیدا است. مشکل overfitting تقریبا حل شده است. و دقت و خطای دادگان آموزش و آزمون به هم نزدیک هستند.

در یک شبکه عصبی بزرگ هنگامی که مدت زیادی مشغول آموزش است. احتمال overfit شدن بالا است. در این حالت مدل برای نویز های موجود در دیتاست آموزش نیز آپدیت می‌شود. و پترن کلی داده ها را یاد نمی‌گیرد. لایه dropout با آرگومان p می‌آید. و در هر لایه به احتمال p هر نورون را خاموش می‌کند. یعنی در مرحله forward, backward شرکت داده نمی‌شود. این کار باعث می‌شود که هنگام هر آپدیت تعدادی نورون خاموش باشند و در نتیجه نویزی که هر نورون در یک مرحله یاد می‌گیرد. نوروی دیگری یاد نگرفته پس تابع هزینه بالا می‌رود و در آپدیت بعدی "احتمالا" نورونی که نویز را یادگرفته برای کاهش تابع هزینه طوری آپدیت می‌شود که نویز را فرا نگیرد. این کار به نوعی غیرهمبسته کردن وزن ها است که توانایی فوق العاده ای روی از بین بردن overfitting دارد.

.4

در یک شبکه عصبی هنگامی که مشغول آموزش است. در ابتدا پترن های کلی دیتا را یاد می‌گیرد و با گذشت epoch های کم کم به سراغ یادگیری نویز های موجود در دیتاست می‌رود تا تابع هزینه خود را کاهش دهد. این کار باعث overfit و عملکرد بد در دیتاست آزمون می‌شود. برای جلوگیری از این اتفاق می‌توان مدل را تا جایی آموزش داد که خطای دادگان آزمون شروع به زیاد شدن کند.(که همان overfit) است. مهمترین معیار استفاده شده در این روش خطای دیتاست آزمون است که تعداد epoch کافی را مشخص می‌کند.

با پیاده سازی Early Stopping و قرار دادن پارامتر های min_delta = -0.1, patience = 3 مشخص می‌کنیم که اگر 3 بار مقدار خطای دیتاست آزمون از خطای مرحله قبلش 0.1 بیشتر شد آموزش را متوقف کن. در این حالت بعد از 7 epoch آموزش متوقف شده و دقت ٪70.10 و خطای 0.8488 بدست آمد.

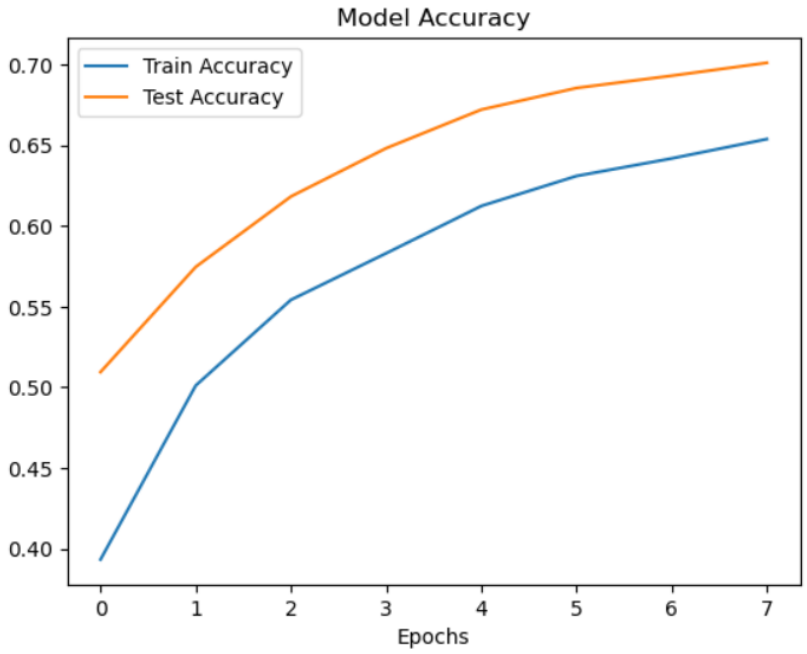

Figure 29 نمودار دقت مدل در روش **early stopping**

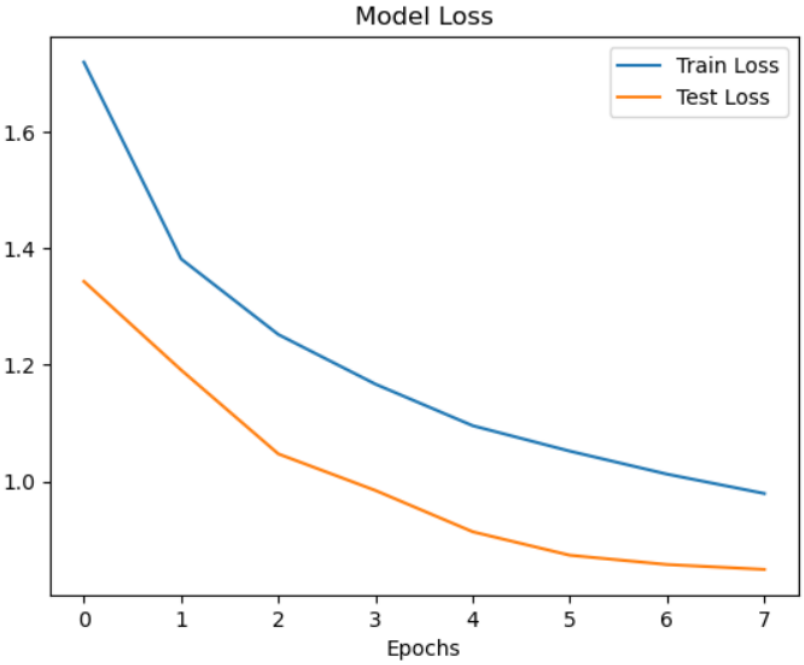

Figure 30 نمودار خطای مدل در روش **early stopping**

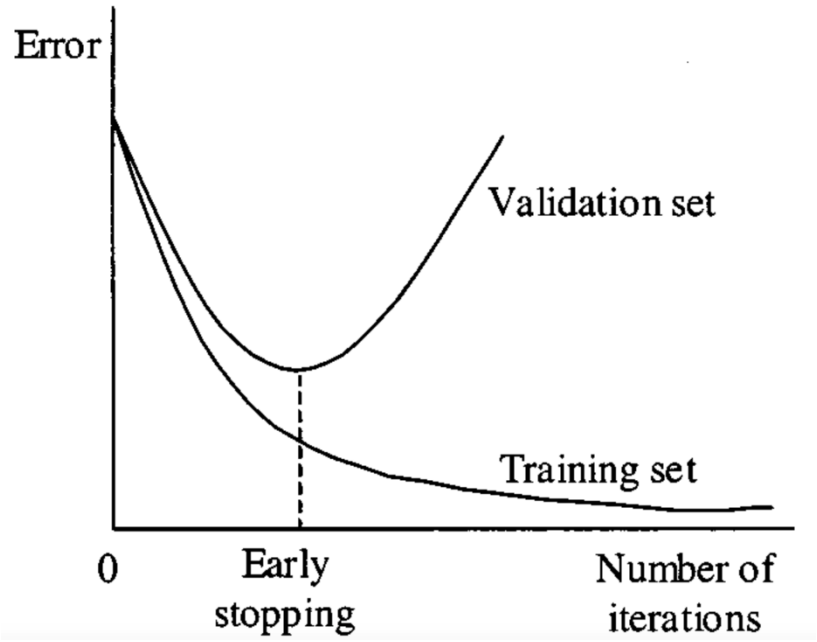

Figure 31 شماتیک توقف زودهنگام

سوال 3

الف) آشنایی با شبکه EfficientNet

الف)

مدل EfficientNetB0 از 18 لایه conv در 6 stage تشکیل شده است. مدل تصاویر 224*224*3 را می‌گیرد. و در لایه های conv با kernel size 3 و 5 طول و عرض آن را کاهش و عمق آن را زیاد می‌کند.

در انتهای لایه های conv هر تصویر به اندازه 7*7*1280 می‌رسد. که به لایه های pooling و fully connected و فعال ساز softmax ختم می‌شود.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 32 ساختار شبکه **EfficeientNetB0** منبع

ب)

مدل EfficientNet 8 نسخه دارد. که تعداد پارامتر های آن از 5.5M شروع شده و تا 66M پیش می‌رود. در همه نسخه ها بلوک اول به اسم stem و بلوک آخر مشترک است.

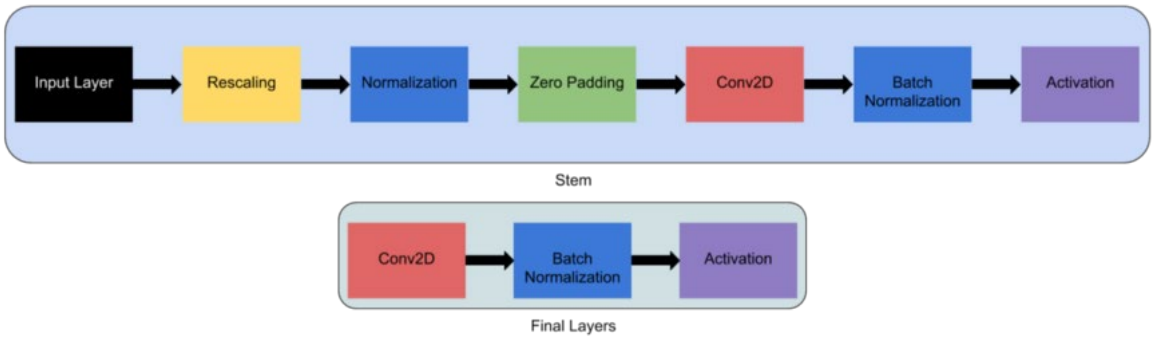

Figure 33 لایه های اول و آخر معماری **EfficientNet**

حال اگر ماژول هایی بصورت زیر تعریف کنیم:

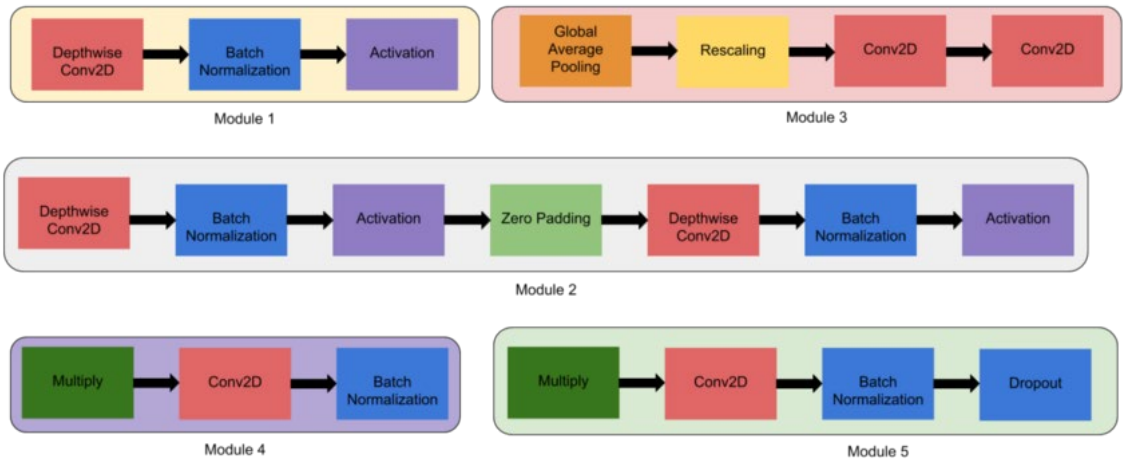

Figure 34 ماژول های مورد استفاده در مدل **EfficientNet**

پس می‌توان معماری های مختلف EfficientNet را بصورت زیر نمایش داد:

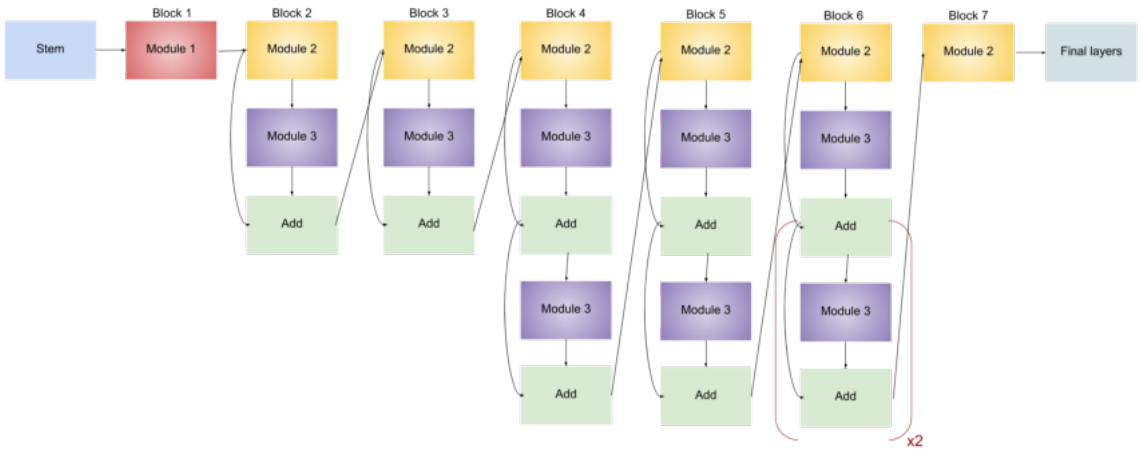

Figure 35 معماری **EfficientNetB0**

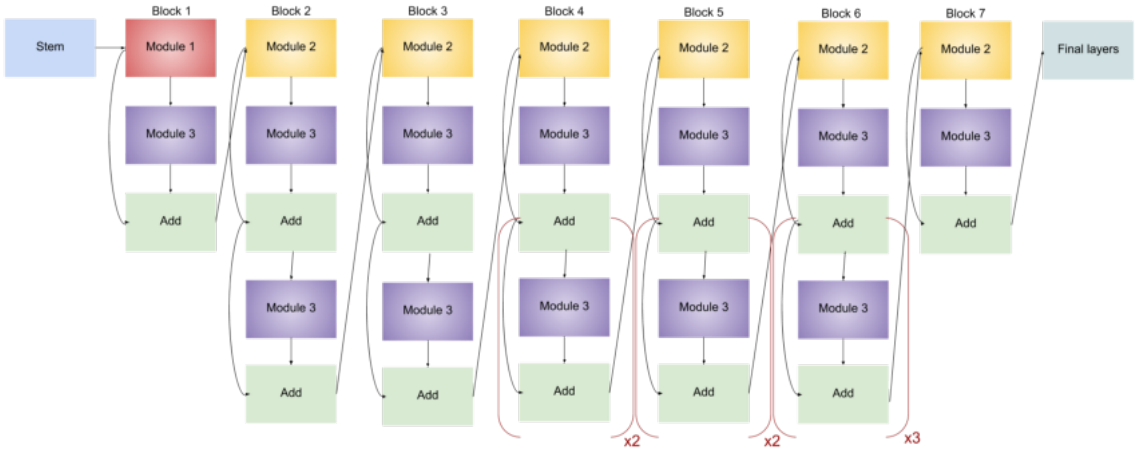

Figure 36 معماری **EfficientNetB1**

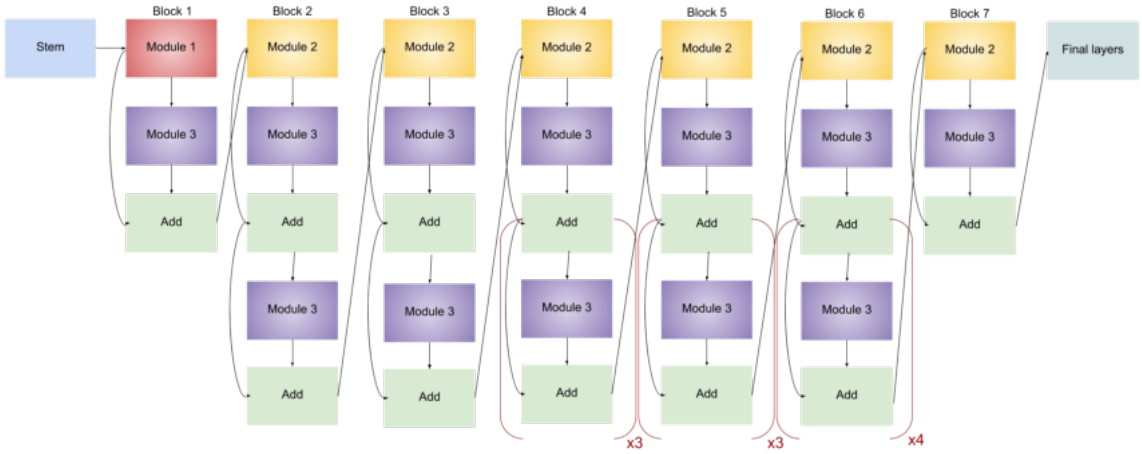

Figure 37 معماری **EfficientNetB3**

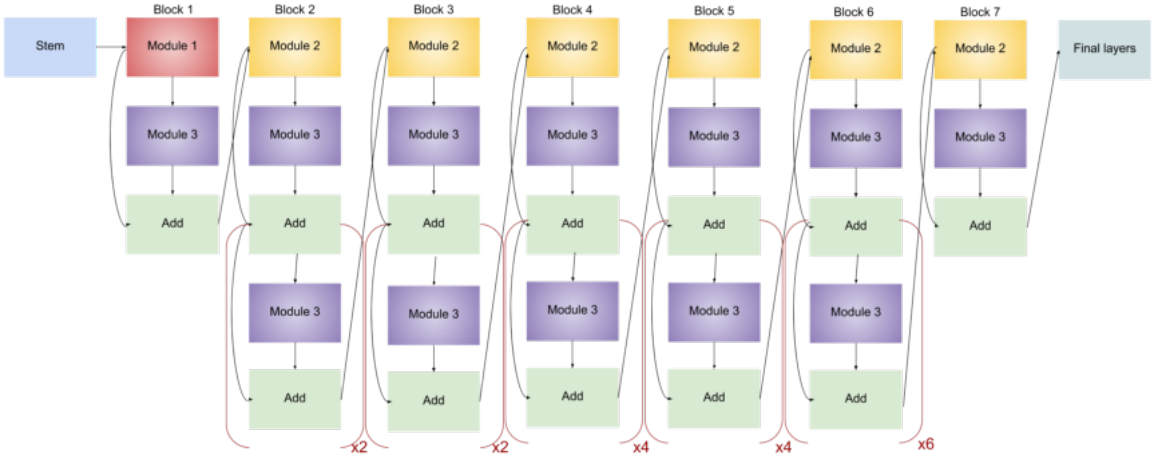

Figure 38 معماری **EfficientNetB4**

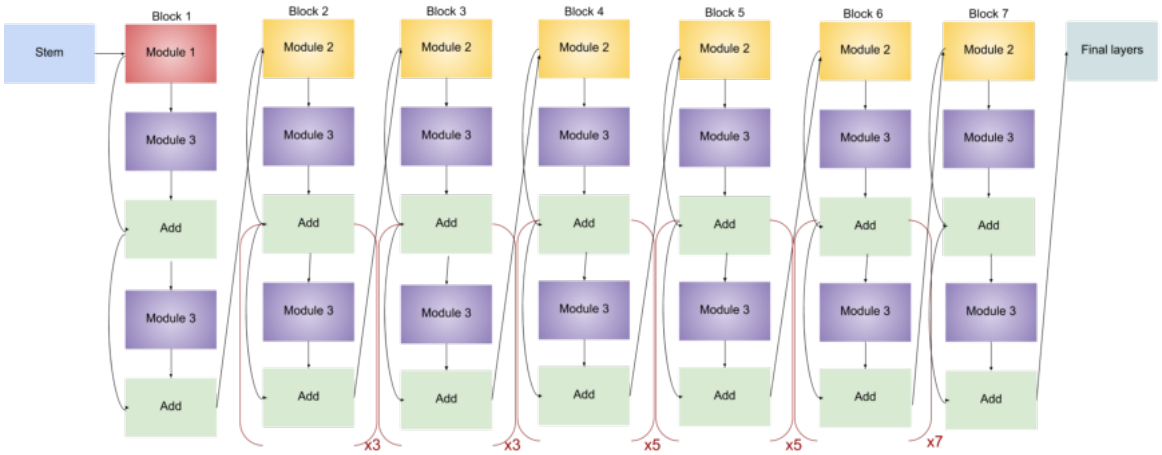

Figure 39 معماری **EfficientNetB5**

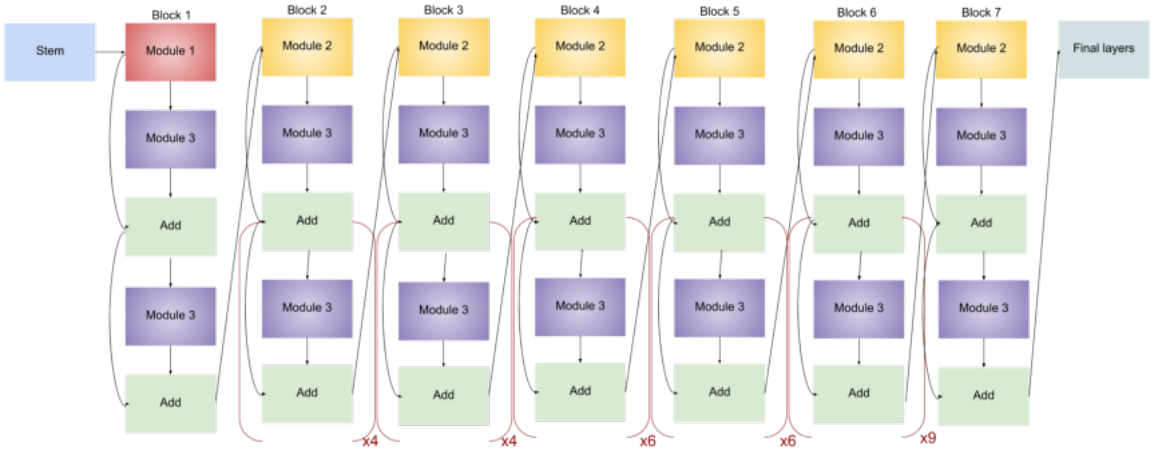

Figure 40 معماری **EfficientNet6**

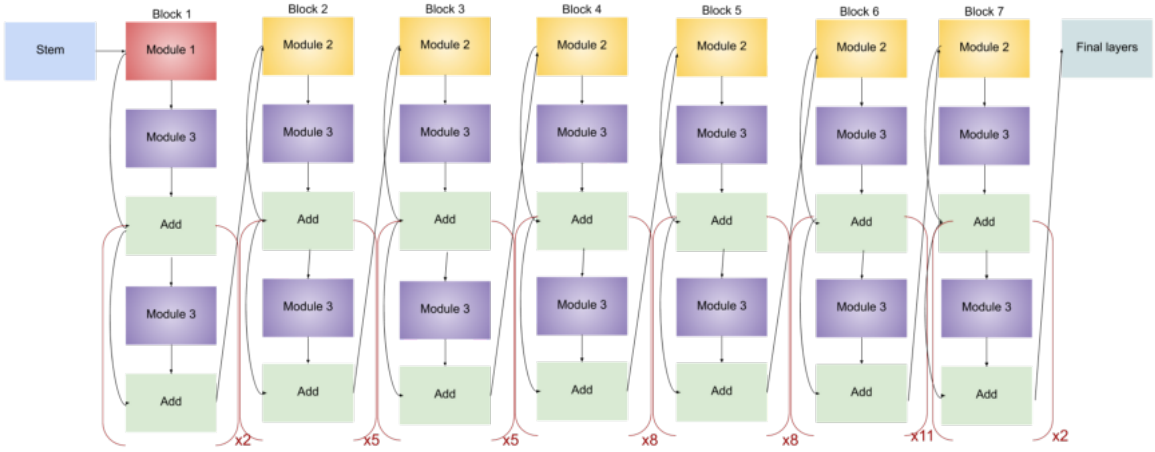

Figure 41 معماری **EfficientNetB7**

ج)

تصویر اولیه برای دادن به کانال باید 224*224 در 3 کانال باشد. (البته میتوان در transfer learning سایز ورودی را مشخص کرد و الزامی به داشتن سایز 224*224 نیست)

د)

این شبکه از بقیه شبکه های موجود همانند resnet, mobilenet, Nasnet, Gpipe بسیار بهینه تر عمل می‌کند. دلیل آن این است که مدلی که ساخته می‌شود. هنگامی که ما نیاز به scale up کردن مدل برای عملکرد بهتر داریم این مدل همه‌ی پارامتر های شبکه اعم از عمق آن، عرض آن و رزولوشن و تعداد کانال را با هم به نسبت مشخص زیاد می‌کند. در حالی که در بقیه شبکه ها فقط یکی از آن ها یا ترکیب غیر بهینه از آن ها زیاد می‌شد. که باعث می‌شد مدل به اشباع برود

ب)پیاده سازی شبکه به کمک ایده Transfer Learning

شبکه را پیاده سازی کرده و به انتهای آن یک لایه با 64 نورون و لایه خروجی با 10 نورون اضافه کردیم. سپس قسمت اضافه شده به مدل به بر روی دیتاست cifar10 آموزش دادیم. به دقت ٪71.63 و خطای 0.1314 رسید. سپس یک عکس را به عنوان تست به مدل دادیم. برای پیش پردازش ابتدا آن را ریسایز به 32*32 کردیم که سایز ورودی مدل بود. سپس transform داده های آموزش را بر روی آن انجام دادیم. عکس دسته جمعی سگ، گربه، خرگوش، طوطی و سوسمار(بسیار شبیه به قوباغه) است. با دادن عکس به مدل 3 دسته با بیشترین احتمال را به ترتیب قوباغه و پرنده و گربه شدند. در یک عکس دیگر که یک سگ و گربه در آن حضور داشتند. مدل به ترتیب سگ و قورباغه و گربه تشخیص داد.

ج) رفع یک مشکل خاص شبکه

اگر تصویر جز اشیایی که قبلا توسط مدل پایه در دیتاست ImageNet باشد. کافی است تعدادی عکس از آن کلاس به دیتاست خود اضافه کرده و آن را مجددا آموزش دهیم. اما اگر یک تصویر خیلی خاص از چیزی که حتی در دیتاست ImageNet نیز وجود نداشت داشتیم. باید تعدادی از لایه های جلویی یادگرفته شده در مدل پایه را مجددا با دیتاست جدید آموزش دهیم. (در لایه های conv هرچه جلو می‌رویم مدل ویژگی های سطح بالا تری از عکس ها را استخراج می‌کند. برای همین لایه های اول که ویژگی هایی مانند خط راست، کج، دایره و ... را استخراج می‌کند. را نگه داشت و لایه های آخر را دوباره آموزش می‌دهیم.

د) آموزش شبکه با مجموعه دادگان جدید

برای آموزش با دیتاست جدید از دیتاست human-horse استفاده می‌کنیم. که شامل 1300 عکس می‌باشد.(توضیح پیش پردازش در پیوست). با دادن عکس ها به مدل و اضافه کردن یک لایه 64 نورون و لایه تک نورون خروجی مدل علیرغم تغییرات بسیار، خروجی قابل قبولی ارائه نمی‌کند.

پیوست

سوال 1)

برای این سوال از کتابخانه ها ی pytorch, NumPy استفاده می‌کنیم.

ورودی، خروجی، وزن ها و بایاس ها را در تنسور ریخته و یک کلاس فرزند از nn.module می‌سازیم. درون این کلاس لایه ها را با وزن ها و بایاس های گفته شده در سوال initialize می‌کنیم. سپس یک instance از توابع فعال ساز ایجاد کرده و در مرحله بعد تابع forward را تعریف می‌کنیم که لایه لایه ورودی و خروجی ها را حساب می‌کند.

یک نمونه از کلاس شبکه ایجاد می‌کنیم. تابع هزینه را mse قرار می‌دهیم و از روش گرادیان نزولی با نرخ یادگیری 0.1 استفاده می‌کنیم. سپس برای اجرای مدل یک بار خروجی را حساب کرده. Loss را حساب کرده و عمل backpropagation را با دستور backward انجام می‌دهیم. در آخر نیز وزن ها را آپدیت کرده و گرادیان را صفر می‌کنیم.

با آپدیت شدن مدل وزن ها را چاپ می‌کنیم. و عمل بروز رسانی را یک بار دیگر نیز انجام می‌دهیم.

سوال 2)

(MLP

ابتدا کتابخانه های لازم را import می‌کنیم. سپس قسمتی از دادگان آموزش را به ارزیابی اختصاص می‌دهیم. لیبل کلاس ها را مشخص کرده و در یک for، 10 تا از نمونه های آموزش را به همراه لیبل آن نمایش می‌دهیم. سپس برای دادن دیتا به شبکه MLP لازم است که عکس های 32*32*3 به وکتور تبدیل شوند که به کمک تابع reshape() انجام شد. در آخر نیز لیبل نمونه ها با دستور to_categorcial بصورت one-hot انکود شدند.

برای ساختن مدل یک نمونه از کلاس keras.layers.sequential می‌سازیم و لایه ها را به آن اضافه می‌کنیم. اندازه لایه اول باید برابر اندازه داده های ورودی 3072 و اندازه لایه آخر باید اندازه تعداد کلاس ها باشد. هر مدل را بر اساس خواسته های سوال طراحی و آموزش می‌دهیم.

(CNN

برای قسمت CNN ابتدا داده ها را به کمک StandardScaler() استاندارد می‌کنیم. سپس یک کلاس از keras.layers.sequential ایجاد می‌کنیم. و لایه ها را به آن اضافه می‌کنیم. اندازه لایه اول باید برابر اندازه داده های ورودی (32,32,3) و اندازه لایه آخر باید اندازه تعداد کلاس ها باشد. هر مدل را بر اساس خواسته های سوال طراحی و آموزش می‌دهیم.

برای قسمت Early Stopping از tf.keras.callbacks() تابع Early Stopping را با آرگومان های گفته شده فرا می‌خوانیم. سپس از این callback در آموزش مدل استفاده می‌کنیم. تا هر وقت که خطای آزمون از حدی بیشتر شد آموزش مدل متوقف شود.

سوال 3)

(2

برای این سوال ابتدا مدل پایه EfficientNetB0 را از قسمت tf.keras.applications وارد می‌کنیم. آرگومان های آن وزن دیتاست imagenet و imput_shape آن (32,32,3) می‌باشد. سپس برای اضافه کردن مدل خود لایه آخر مدل پایه را در یک متغیر ریخته سپس به آن همانند ورودی یک شبکه برخورد می‌کنیم. ابتدا ورودی را flatten کرده تا بصورت یک بردار درآید. سپس یک لایه با 64 نورون به آن اضافه می‌کنیم. بعد لایه BN و لایه dropout را اضافه کرده و لایه خروجی را به تعداد کلاس ها 10 نورون می‌گذاریم.

مدل را به کمک تابع keras.models.Model ساخته و کامپایل می‌کنیم. سپس آن را در 10 دوره اجرا می‌کنیم.(دیتاست همانند سوال 2 قسمت CNN استاندارد شدند)

برای قسمت دوم 2 عکس را به کمک کتابخانه cv2 وارد می‌کنیم. و آن را ریسایز کرده و ترتیب کانال های رنگی آن را تغییر می‌دهیم. سپس transform استاندارد یادگرفته شده در قسمت قبل را روی آن اعمال می‌کنیم. و به مدل می‌دهیم. مدل یک آرایه 10 تایی برمی‌گرداند هر هر درایه احتمال کلاس مربوطه را نشان می‌دهد. 3 کلاس با بیشترین مقدار را برای هر 2 عکس چاپ می‌کنیم.

(3

این قسمت تقریبا مشابه قسمت قبل است. با این تفاوت که در مدل قبلی همه وزن های لایه های مدل پایه فریز شده بودند. اما در این قسمت برای این که مشکل شرح داده شده حل شود. وزن تعدادی از لایه های ماقبل آخر را unfreeze کرده تا در مرحله آموزش، آپدیت شوند.

4)

برای قسمت آخر سوال ابتدا دیتاست human-horse را از سایت گوگل دانلود کرده سپس آن را unzip کرده و در پوشه هایی به همان اسم می‌ریزیم. برای این سوال از data augmentation استفاده می‌کنیم. که تابع ImageDataGenerator آرگومان هایی برای rescale, flip, shift, rotation دارد. سپس از تابع flow_from_directory استفاده می‌کنیم. که دیتاست را که در آدرسی از حافظه است. بصورت batch, batch در رم لود کرده و به مدل بدهد.(این کار برای دیتاست هایی که اندازه در چند گیگابایت دارند ضروری است و نمی‌توان همه آن را همزمان در رم لود کرد)

سپس برای نمونه یک عکس از دیتاست را به کمک cv2.imread() نمایش می‌دهیم. ادامه فراید ساخت و آموزش مدل همانند قسمت 2 سوال است. با این فرق که مدل 2 کلاسه است. و از تابع هزینه binary_crossentropy استفاده می‌کنیم.