

به نام خدا

پروژه امتیازی درس یادگیری ماشین

مهدی نیک نژاد ۴۰۲۱۳۰۴

فهرست مطالب

1 سوال ۱
1 Data loading and preprocessing
2 model::Transformer
2 MultiHeadAttention
3 FeedForward
4 model::iTransformer
9 سوال ۲

سوال ۱

Untitled 14.ipynb

Data loading and preprocessing

ابتدا یک کلاس تعریف می کنیم برای load کردن و پیش پردازش داده ها. در این بخش split را برابر با ۰.۸ می گذاریم. و در ادامه len و getitem را هم تعریف می کنیم.

```
class ExchangeRateDataset(Dataset):
    def __init__(self, data_path, seq_len=96, pred_len=96, train=True):
        self.seq_len = seq_len
        self.pred_len = pred_len
        df = pd.read_csv(data_path)
        df = df.drop(columns=["date"])
        self.scaler = StandardScaler()
        data = self.scaler.fit_transform(df.values) # shape:
[num_samples, 8]
        split = int(len(data) * 0.8)
        self.data = data[:split] if train else data[split:]
```

```

def __len__(self):
    return len(self.data) - self.seq_len - self.pred_len + 1

def __getitem__(self, index):
    x = self.data[index : index + self.seq_len]      # [seq_len, 8]
    y = self.data[index + self.seq_len : index + self.seq_len +
self.pred_len] # [pred_len, 8]
    return torch.tensor(x, dtype=torch.float32), torch.tensor(y,
dtype=torch.float32)

```

model::Transformer

در این بخش اجزای مدل transformer را پیاده سازی می کنیم که شامل چند بخش است:

- MultiHeadAttention
- FeedForward

MultiHeadAttention

این مکانیزم می است که به مدل اجازه می دهد تا به طور همزمان روی قسمت های مختلف دنباله ورودی تمرکز کند. و خود شامل چند بخش است:

- تبدیل ورودی X به Q, K, V با استفاده از چندین linear layer
- محاسبه attention score با استفاده از dot product بین Q, K
- اعمال softmax به scores برای رسیدن به attention weights
- استفاده از وزن ها برای محاسبه wighted sum of values یعنی V
- خروجی نهایی به یک لایه خطی دیگر پاس داده می شود.

```

class MultiHeadAttention(nn.Module):
    def __init__(self, d_model, n_heads):
        super().__init__()
        assert d_model % n_heads == 0, "d_model must be divisible by
n_heads"
        self.d_head = d_model // n_heads
        self.n_heads = n_heads
        self.query = nn.Linear(d_model, d_model)
        self.key = nn.Linear(d_model, d_model)
        self.value = nn.Linear(d_model, d_model)
        self.fc_out = nn.Linear(d_model, d_model)

    def forward(self, x, mask=None):
        B, L, D = x.shape
        Q = self.query(x).view(B, L, self.n_heads,
self.d_head).transpose(1,2)
        K = self.key(x).view(B, L, self.n_heads,
self.d_head).transpose(1,2)

```

```

        V = self.value(x).view(B, L, self.n_heads,
self.d_head).transpose(1,2)
        scores = torch.matmul(Q, K.transpose(-2,-1)) /
math.sqrt(self.d_head)
        if mask is not None:
            scores = scores.masked_fill(mask==0, float('-inf'))
        attn = F.softmax(scores, dim=-1)
        out = torch.matmul(attn, V)
        out = out.transpose(1,2).contiguous().view(B, L, D)
        return self.fc_out(out)

```

نکات:

d_{model} : بعد وکتور ورودی و خروجی

n_{heads} : تعداد سرهای توجه. هر سر روی بخشی از بعد ورودی عملیات انجام می دهد. پس باید بخشپذیر بر آن باشد.

وکتور ورودی این ابعاد را دارد. (B, L, D)

model dimension (d_{model}): D

sequence length (# of timesteps): L

batch size: B

FeedForward

```

class FeedForward(nn.Module):
    def __init__(self, d_model, d_ff, dropout=0.1):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(d_model, d_ff),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(d_ff, d_model)
        )
    def forward(self, x):
        return self.net(x)

```

در ادامه از این بخشهایی که پیاده سازی شدند برای بلوک transformer استفاده می کنیم.

```

class TransformerBlock(nn.Module):
    def __init__(self, d_model, n_heads, d_ff, dropout=0.1):
        super().__init__()
        self.attention = MultiHeadAttention(d_model, n_heads)
        self.norm1 = nn.LayerNorm(d_model)
        self.ffn = FeedForward(d_model, d_ff, dropout)
        self.norm2 = nn.LayerNorm(d_model)
        self.dropout = nn.Dropout(dropout)

```

```

def forward(self, x):
    attn_out = self.attention(x)
    x = self.norm1(x + self.dropout(attn_out)) # Add and Norm
    ffn_out = self.ffn(x)
    return self.norm2(x + self.dropout(ffn_out)) # Add and Norm

```

```

class Transformer(nn.Module):
    def __init__(self, seq_len, pred_len, num_features=8, d_model=128,
n_heads=4, d_ff=256, num_layers=2, dropout=0.1):
        super().__init__()
        self.seq_len = seq_len
        self.pred_len = pred_len
        # Project the 8 features at each time step to d_model
        self.input_proj = nn.Linear(num_features, d_model)
        self.transformer_blocks = nn.ModuleList(
            [TransformerBlock(d_model, n_heads, d_ff, dropout) for _ in
range(num_layers)]
        )
        self.fc_out = nn.Linear(d_model, num_features)

    def forward(self, x):
        # x: [B, seq_len, num_features]
        x = self.input_proj(x) # [B, seq_len, d_model]
        for block in self.transformer_blocks:
            x = block(x)
        last = x[:, -1, :] # [B, d_model]
        # For forecasting, we take the last time-step representation and
repeat it pred_len times, then map back to 8 features.
        repeated = last.unsqueeze(1).repeat(1, self.pred_len, 1) # [B,
pred_len, d_model]
        out = self.fc_out(repeated) # [B, pred_len, num_features]
        return out

```

model::iTransformer

در ادامه برای مدل iTransformer، یک کلاس تعریف میکنیم که هر سری زمانی را به یک توکن نگاه کند. هدف اصلی این کلاس تعبیه ورودی داده های سری زمانی در یک فضای با ابعاد بالاتر (d_model) است.

```

# DataEmbedding_inverted: embeds the (inverted) time-series.
class DataEmbedding_inverted(nn.Module):
    def __init__(self, c_in, d_model, dropout=0.1):
        super(DataEmbedding_inverted, self).__init__()
        self.value_embedding = nn.Linear(c_in, d_model)
        self.dropout = nn.Dropout(p=dropout)

```

```
def forward(self, x, x_mark):
    # x: [B, T, N] where T = seq_len, N = number of variates.
    # Inversion: treat T as channels.
    x = x.permute(0, 2, 1) # now: [B, N, T]
    # (If x_mark is provided, one could concatenate; here we assume
    None.)
    x = self.value_embedding(x) # [B, N, d_model]
    return self.dropout(x)
```

c_in: تعداد variate ها (فیچرها)

تبدیل با استفاده از یک لایه fully connected layer برای project کردن داده های سری زمانی اصلی به یک فضای جدید انجام می شود.

در بخش forward داریم:

ورودی X اندازه (B,T,N) دارد. batch size: B sequence length: T # of variates: N

این ورودی X یک timeseries را نشان می دهد با N فیچر در طول T گام زمانی.

بعد از permute(0,2,1)، اندازه خروجی برابر می شود با: (B, N, T)

این طوری T به عنوان تعداد کانال ها می باشد و حالا مدل هر variate را به طور جداگانه پردازش می کند.

در آخر ورودی تبدیل شده را به لایه value_embedding می دهیم. این لایه N variates را به فضای d_model، project می کند. و

خروجی می شود: (B, N, d_model)

در ادامه مدل iTransformer مشخص می شود.

```
class iTransformer(nn.Module):
    def __init__(self, configs):
        super(iTransformer, self).__init__()
        self.seq_len = configs.seq_len
        self.pred_len = configs.pred_len
        self.output_attention = configs.output_attention
        self.use_norm = configs.use_norm

        # Embedding: transpose and embed the series into variate tokens
        self.enc_embedding = DataEmbedding_inverted(configs.seq_len,
        configs.d_model, configs.embed, configs.freq, configs.dropout)

        # Use MultiHeadAttention instead of FullAttention
        self.attention = MultiHeadAttention(configs.d_model,
        configs.n_heads) # Change to MultiHeadAttention
```

```

        self.projector = nn.Linear(configs.d_model, configs.pred_len,
bias=True) # For forecasting

    def forecast(self, x_enc, x_mark_enc, x_dec, x_mark_dec):
        if self.use_norm:
            means = x_enc.mean(1, keepdim=True).detach()
            x_enc = x_enc - means
            stdev = torch.sqrt(torch.var(x_enc, dim=1, keepdim=True,
unbiased=False) + 1e-5)
            x_enc /= stdev

        # Apply embedding
        enc_out = self.enc_embedding(x_enc, x_mark_enc)

        # Use MultiHeadAttention for the encoding part (as per pseudocode)
        attention_out = self.attention(enc_out) # Apply attention
directly

        # Project the output to the required shape for forecasting
        dec_out = self.projector(attention_out).permute(0, 2, 1)[:x_enc.shape[-1]]

        if self.use_norm:
            dec_out = dec_out * (stdev[:, 0, :].unsqueeze(1).repeat(1,
self.pred_len, 1))
            dec_out = dec_out + (means[:, 0, :].unsqueeze(1).repeat(1,
self.pred_len, 1))

        return dec_out

    def forward(self, x_enc, x_mark_enc, x_dec, x_mark_dec, mask=None):
        dec_out = self.forecast(x_enc, x_mark_enc, x_dec, x_mark_dec)
        return dec_out[:, -self.pred_len:, :] # [B, pred_len, N]

```

تابع forecast یک دنباله ورودی می گیرد و یک پیش بینی انجام می دهد. عملاً ZSCORE محاسبه می شود. و به طور کلی:

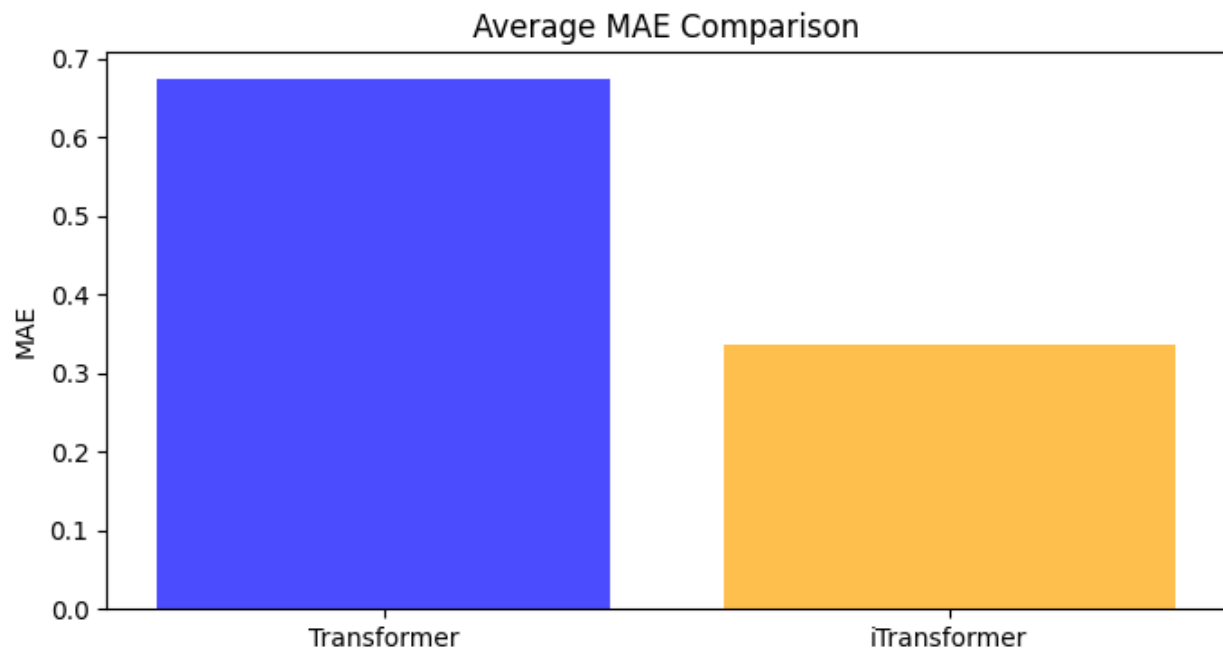
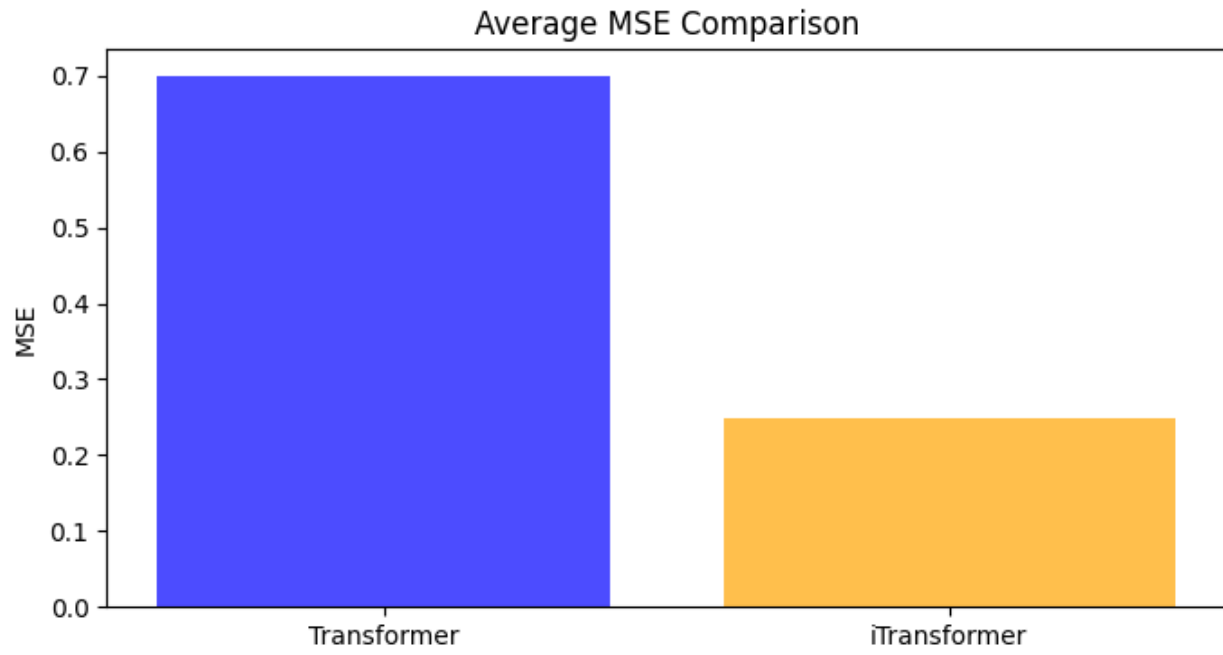
- Normalization
- Converts the input into the **embedded variate tokens**
- **Applies multi-head self-attention across features**
- Projection

و ورودی تابع forward برابر است با این موارد:

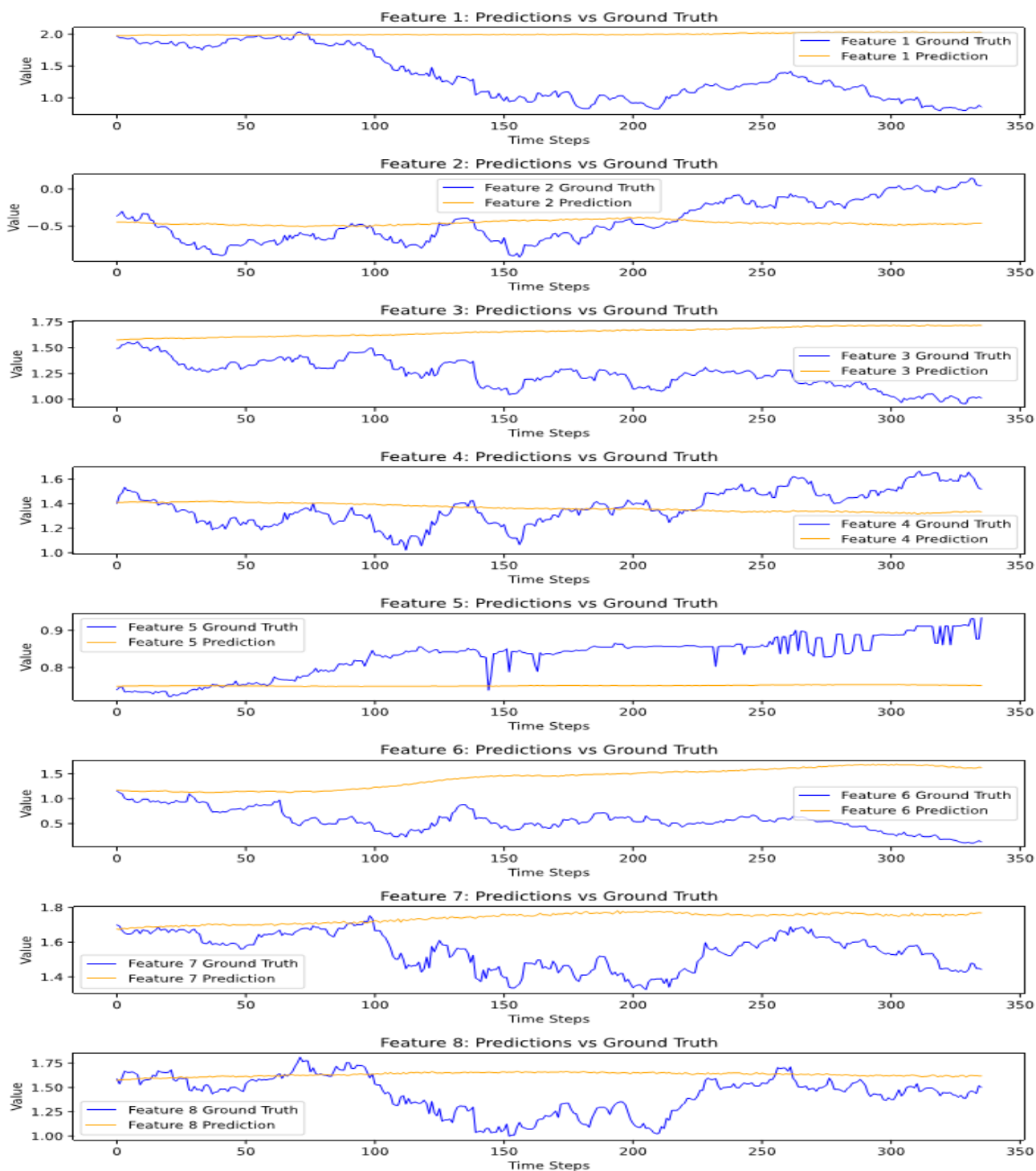
X_enc: Past time series data (input to the encoder).

time-related information for the past data : X_mark_enc

در ادامه train_model برای هر دو مدل را نوشتیم. و سپس evaluation را برای هر دو مدل نوشتیم. و در آخر مدل را آموزش دادیم و این هم حاصل میانگین MSE و MAE برای هر دو مدل با prediction_length های مختلف را می آوریم:



و در آخر هم می خواستیم خروجی کار را روی دیتاست exchange_rate ببینیم (با pred_len=336) و این حاصل است که اصلا خوب نیست و علت را نمی دانم.

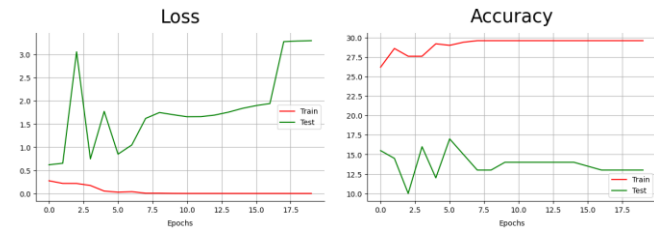
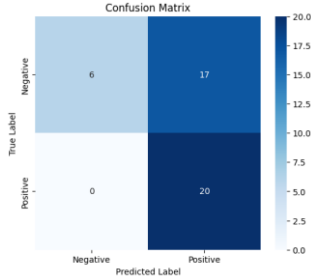
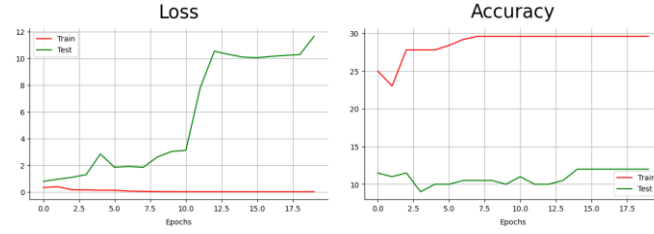
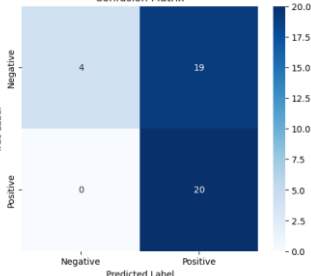
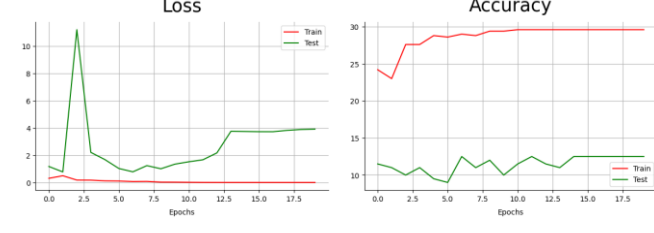
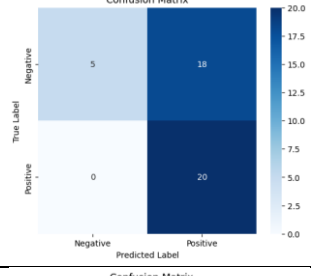
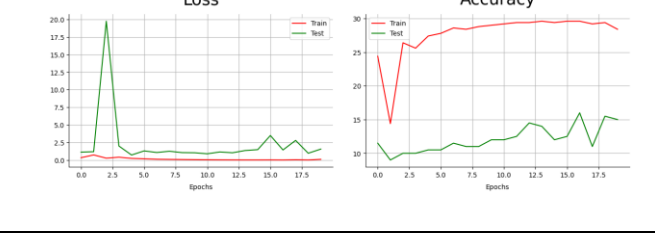
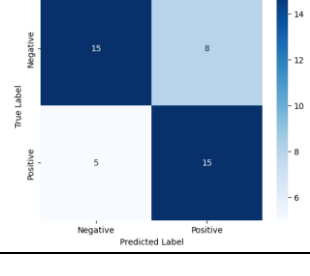


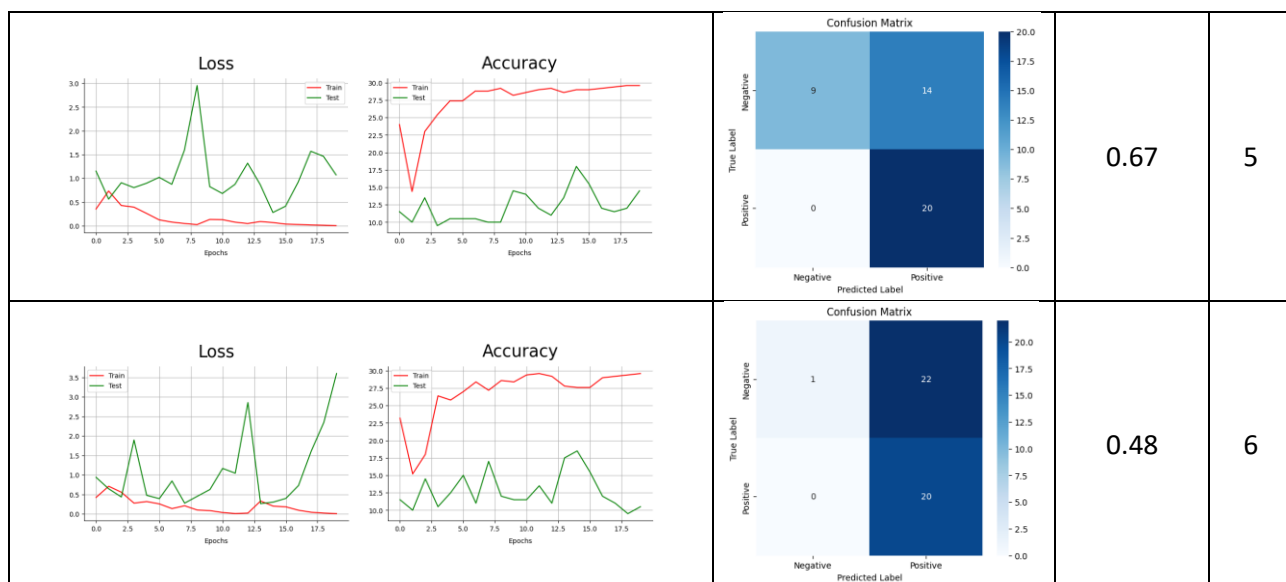
سوال ۲

Untitled 8.ipynb

Data augmentation

برای دیتا آگمنتیشن ۴ مورد را استفاده کردیم. (از transforms)

Accuracy and Loss	Confusion matrix	Accuracy	# of conv layers
		0.6	1
		0.55	2
		0.58	3
		0.69	4



توجه: هم برای داده های آموزش و هم برای داده های آزمون، تعداد لایه های مختلف کانولوشن استفاده شد و در جدول آمده است. (برای تعداد epoch=20)

متأسفانه نتایج خوب نشدند علی رغم اینکه به نظرم کد را درست نوشته ام.

پایان