Ruslan Osmanov ID: 327480026

Beni Wolftson ID: 316535798

Please for correct readings of saved images use Jupyter Notebook

If images didnot show ,you can see them if you open notebook in Git

Logs of training ,report in pdf format and notebook also storen in git

Link to Git : https://github.com/deamon312/Deep_Learning-Computer_Vision/tree/main/Project%202

Import Dependencies

In [1]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report ,ConfusionMatrixDisplay ,confusion_matrix
from keras import layers
from tensorflow.keras.callbacks import TensorBoard
sns.set_theme(style='white')
import datetime
import pandas as pd
import random
```

If you are not using Desktop GPUs, skip next cell

In [2]:
```
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
config = ConfigProto()
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)
```

Load Data Set

Let's load pre-splited data from the bloodmnist data set ,that spliten into validation ,training ,and test

In [3]:
```
def load_data(path):
    with np.load(path) as file:
        x_train, y_train = file['train_images'], file['train_labels']
        x_test, y_test = file['test_images'], file['test_labels']
        x_val, y_val = file['val_images'], file['val_labels']
        return (x_train, y_train), (x_test, y_test) , (x_val, y_val)

(x_train, y_train), (x_test, y_test),(x_val ,y_val) = load_data('bloodmnist.npz')
```

Data Set Description

Now we describe our data set dimensions and splits the sets

In [4]:
```
print('Image Dim: {} x {}'.format(x_train.shape[1],  x_train.shape[2]))
print('Training: {} / Validation: {} / Test: {} /Total: {}'.format(x_train.shape[0],x_val.shape[0],x_test.shape[0],np.sum([x
```

```
Image Dim: 28 x 28
Training: 11959 / Validation: 1712 / Test: 3421 /Total: 17092
```

New Split to get overfit

In [5]:
```
# Training Data was reduced to 30%
```

In [6]:
```
X_train, _, y_train, _ = train_test_split(x_train, y_train, train_size=0.3, random_state=55)
```

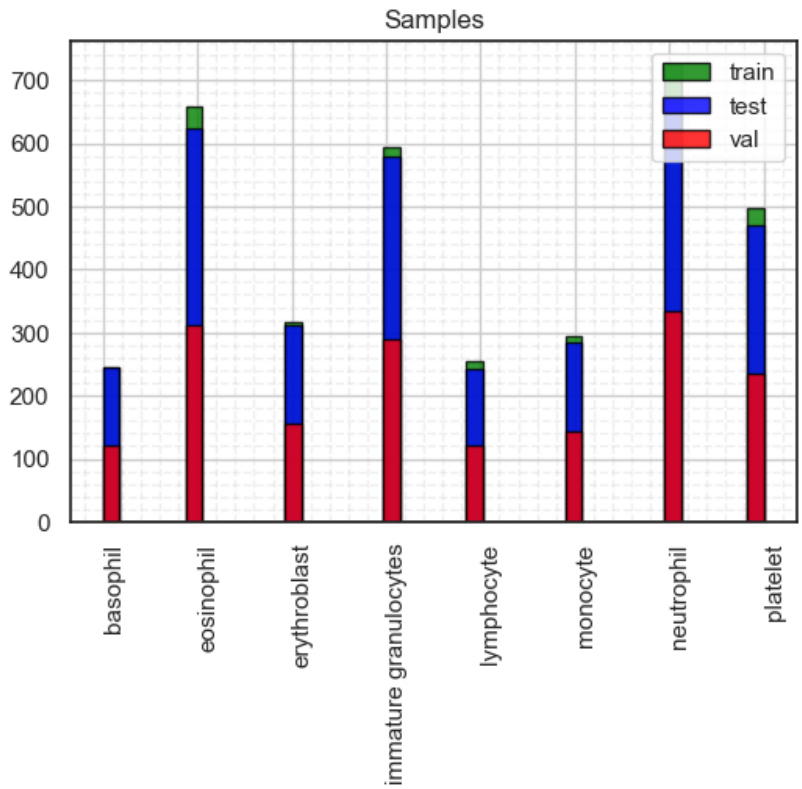
In [7]:
```
print('Training: {} / Validation: {} / Test: {} /Total: {}'.format(X_train.shape[0],x_val.shape[0],x_test.shape[0],np.sum([X
```

```
Training: 3587 / Validation: 1712 / Test: 3421 /Total: 8720
```

In the next plot it can been see an unbalanced classes ,because each class has different amount of samples

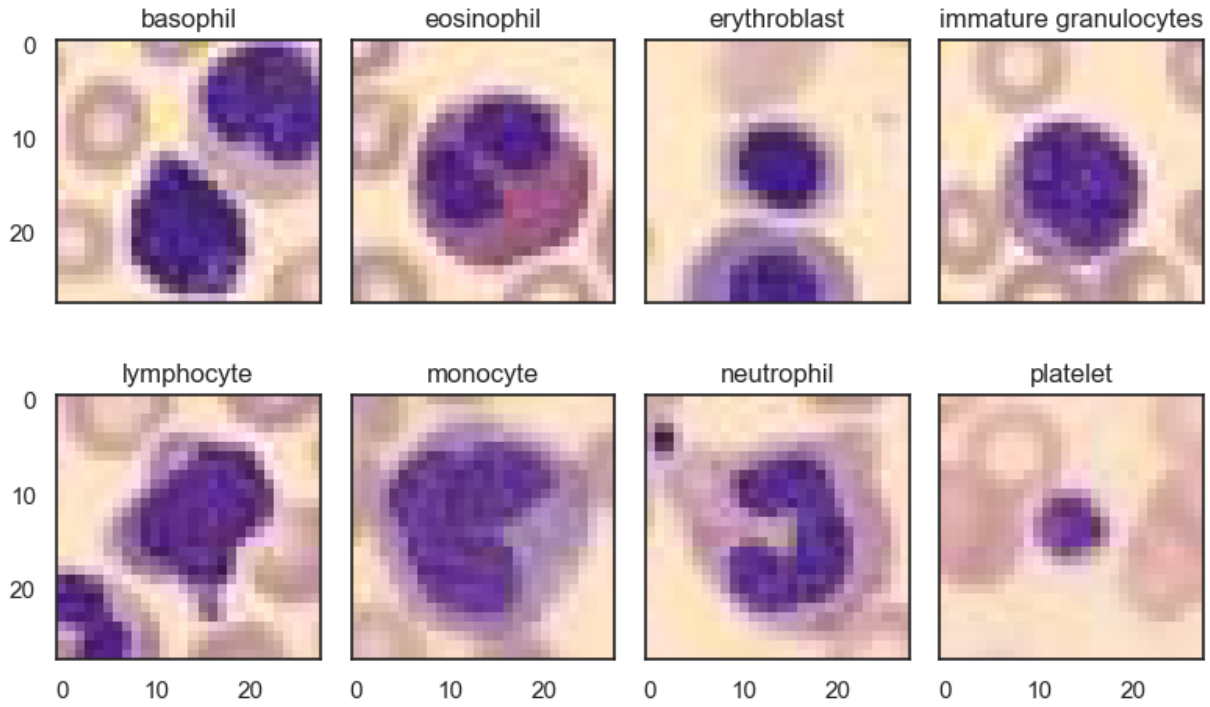
In [8]:
```
fig, ax = plt.subplots(figsize=(6,4))

class_names=["basophil","eosinophil","erythroblast","immature granulocytes","lymphocyte","monocyte","neutrophil","platelet"]
c= ['green','blue','red']
```

```
label = ['train','test','val']

for idx ,a in enumerate([y_train, y_test,y_val]):
    ax.hist(a,color=c[idx],bins=40 ,label=label[idx],edgecolor='black',alpha=0.8)
    ax.grid(visible=True ,which='minor',linestyle='--',alpha=0.3)
    ax.minorticks_on()
    ax.grid()
plt.xticks(ticks=range(8), labels=class_names, rotation=90, horizontalalignment='left')
plt.legend()
plt.title('Samples')
plt.show()
```



Samples from the Training data set

In [9]:
```
# plot first few images
valueList, indicesList = np.unique(y_train, return_index=True)
fig, ax = plt.subplots(2,4,figsize=(8,5),sharex=True ,sharey=True)
for i,idx in enumerate(indicesList):
    if i <=3:
        ax[0,i].imshow(X_train[idx])
        ax[0,i].set_title(class_names[i])
    else:
        ax[1,i-4].imshow(X_train[idx])
        ax[1,i-4].set_title(class_names[i])

plt.tight_layout()
plt.show()
```



In [10]:
```
# Plotting Functions
def evaluation_plots(classifier,y_lim =[0,1.4]):
    losses = pd.DataFrame(classifier.history.history)
    fig,axs =plt.subplots(1,2,figsize=(12,4))
    axs[0].plot(losses[['loss','val_loss']],label=['Train_loss' ,'Val_loss'])
    axs[1].plot(losses[['accuracy','val_accuracy']],label=['Train_accuracy' ,'Val_accuracy'])
    title = ['loss','accuracy']
    for i in range(2):
        axs[i].grid(visible=True ,which='minor',linestyle='--',alpha=0.3)
        axs[i].minorticks_on()
        axs[i].grid()
        axs[i].set_xlabel('epocs')
        axs[i].set_title(title[i])
        axs[i].legend(loc='upper left')
    axs[0].set_ylim(y_lim)
    plt.show()

class estimator: # for purpose of using  metrics from sklern package
    _estimator_type = ''
    classes_=[]
    def __init__(self, model, classes):
        self.model = model
        self._estimator_type = 'classifier'
        self.classes_ = classes
    def predict(self, X):
        y_prob= self.model.predict(X)
        y_pred = y_prob.argmax(axis=1)
        return y_pred
```

- Data normalization is a preprocessing step that scales the values in the data to a specific range [0,1]. It is used to scale the features of a dataset so that they have a similar scale and can be more easily compared.

In [11]:
```
prepare = tf.keras.Sequential([
tf.keras.layers.Rescaling(1./255)])
```

Overfitted models as is

Overfitting

- Overfitting occurs when a machine learning model is trained too well on the training data, and as a result, it does not generalize well to new, unseen data. In other words, the model has learned patterns in the training data that do not hold true for the broader population, and as a result, its predictions on new data are less accurate.

In [12]:
```
classifier = keras.Sequential(name='CNN-128-MP-64-MP-64-MP-Flaten-FC64')
prepare,
classifier.add(layers.Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu',input_shape=(28,28,3)))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(3, 3),padding='same', activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(2, 2), activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Flatten())
classifier.add(layers.Dense(64, activation='relu'))
```

```
classifier.add(layers.Dense(8, activation='softmax'))
classifier.summary()
```

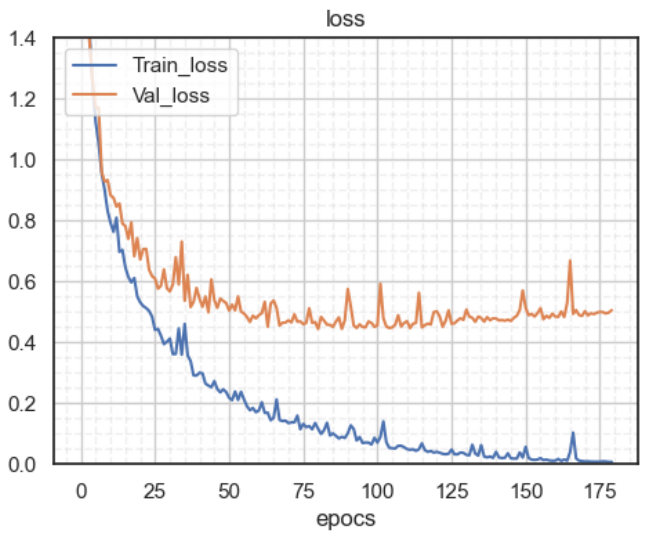
```
Model: "CNN-128-MP-64-MP-64-MP-Flaten-FC64"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 128)       3584

 max_pooling2d (MaxPooling2D  (None, 14, 14, 128)      0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        73792

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 6, 6, 64)          16448

 max_pooling2d_2 (MaxPooling  (None, 3, 3, 64)         0
 2D)

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 64)                36928

 dense_1 (Dense)             (None, 8)                 520

=================================================================
Total params: 131,272
Trainable params: 131,272
Non-trainable params: 0
_________________________________________________________________
```

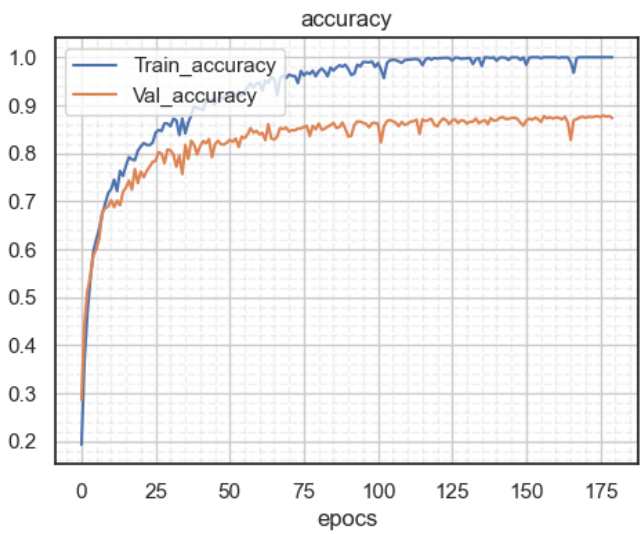
In [13]:
```
log_dir = "logs/CNN/"+'CNN-128-MP-64-MP-64-MP-Flaten-FC64'
tb_callback = TensorBoard(log_dir=log_dir, histogram_freq=1,write_images=True)
```

In [14]:
```
classifier.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.00005),
              metrics=['accuracy'])
```

In [ ]:
```
history=classifier.fit(X_train, y_train, batch_size=64, epochs=180, validation_data=(x_val,y_val),callbacks=[tb_callback])
```

In [ ]:
```
evaluation_plots(classifier)
```





As we can see here ,there are an overfit ,because the model is trained too well on the training data

Augmentation

- Image augmentation is a technique used to artificially increase the size of a training dataset by creating modified versions of images in the dataset. This is done by applying a set of predefined augmentation techniques such as rotation, cropping, scaling, and flipping to the original images. The goal of image augmentation is to introduce variations in the training dataset so that the model can learn to recognize and classify images better, even if they are slightly different from the images in the original dataset. This can help to reduce overfitting, which occurs when a model performs well on the training data but poorly on new, unseen data.

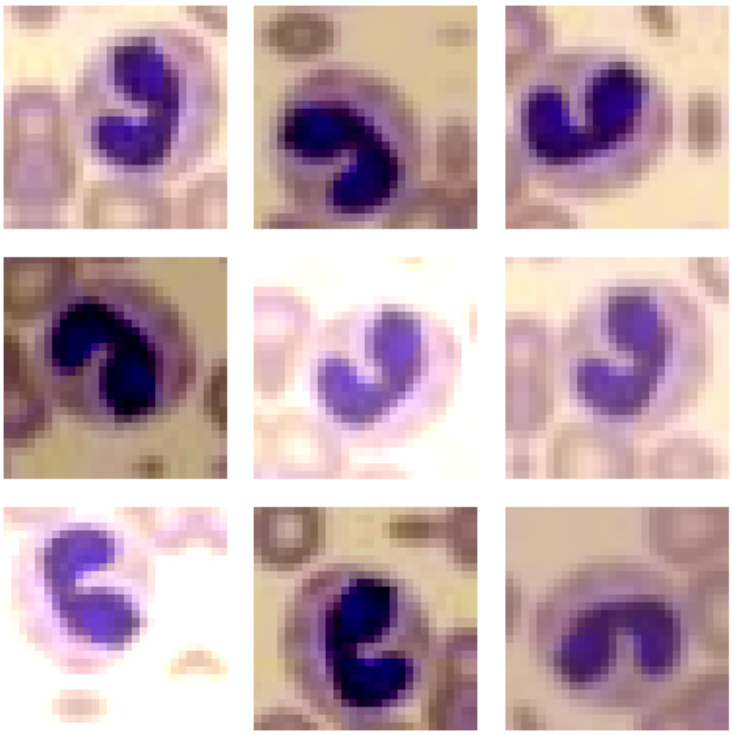
Define an augmentation layer that will be applyed on each image

In [14]:
```
data_augmentation = tf.keras.Sequential([
  layers.Rescaling(1./255) ,
  layers.RandomRotation(0.4),
```

```
  layers.RandomFlip(),
  layers.RandomTranslation(0.15,0.15),
  layers.RandomContrast(0.3),
  layers.RandomBrightness(0.3,value_range=(0, 1)),
])
```

- Plot sample of images after augmentation layer

In [15]:
```
plt.figure(figsize=(5, 5))
for i in range(9):
    augmented_image = data_augmentation(x_train[240])
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_image,vmin=0,vmax=1)
    plt.axis("off")
plt.tight_layout()
```



As can be seen the same image was rotated ,translated,fliped and with different : contrast ,brightnes

Lets apply augmentation layer in our previos model

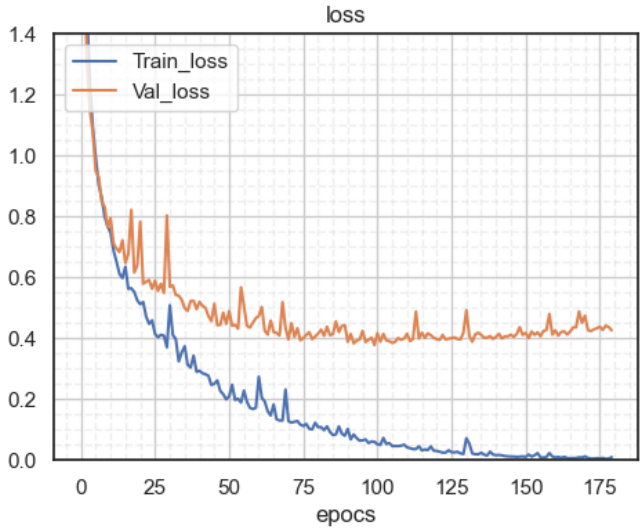
In [77]:
```
classifier = keras.Sequential(name='CNN-Aug-128-MP-64-MP-64-MP-Flaten-FC64')
data_augmentation,
classifier.add(layers.Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu',input_shape=(28,28,3)))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(3, 3),padding='same', activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(2, 2), activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Flatten())
classifier.add(layers.Dense(64, activation='relu'))
classifier.add(layers.Dense(8, activation='softmax'))
```

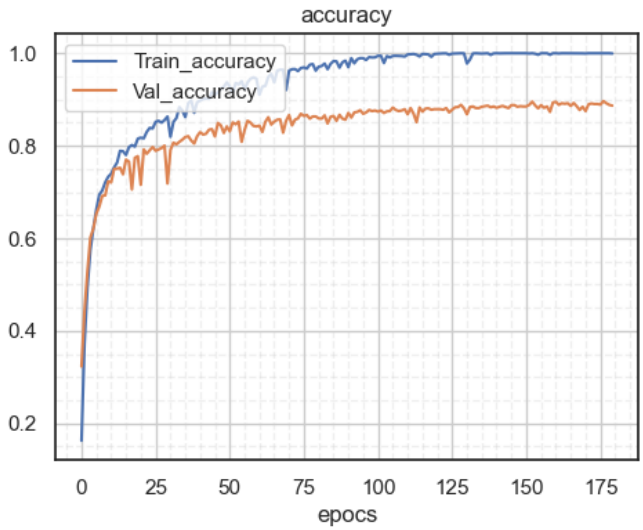
In [78]:
```
log_dir = "logs/CNN/"+'CNN-Aug-128-MP-64-MP-64-MP-Flaten-FC64'
tb_callback = TensorBoard(log_dir=log_dir, histogram_freq=1,write_images=True)
```

In [79]:
```
classifier.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.00005),
              metrics=['accuracy'])
```

In [ ]:
```
history=classifier.fit(X_train, y_train, batch_size=64, epochs=180, validation_data=(x_val,y_val),callbacks=[tb_callback])
```

In [ ]:
```
evaluation_plots(classifier)
```





As we can see here ,there are also overfit but less than previos and learning curve also more smooth moreover accuracy increased

Transfer learning

- Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task. It can greatly save on time and resources, as well as improve the performance of the model on the second task.
- There are a few different ways to use transfer learning in deep learning. One approach is to use the weights from a pre-trained model as the starting point for training a new model on a second task. Another approach is to fine-tune a pre-trained model on the second task by continuing to train it with new data.
- Transfer learning is often used in practice because it can be difficult and time-consuming to train a deep learning model from scratch on a large dataset. By using a pre-trained model as a starting point, it is possible to train a good model much more quickly, and with less data.
- In our aproach we will use Vgg16 pretrained models and only re-train the last and first layers

- Input shape of this models need to be at least 32 X 32 ,in the next step we will re-size and re -scale our data sets to meet the creterions of the VGG16 Pre-trained model.
- We dont want to re-size it to bigger sizes ,because our images are small 28 X 28 and re -size to bigger can couse a negative effect

In [14]:
```
IMAGE_SIZE =32
```

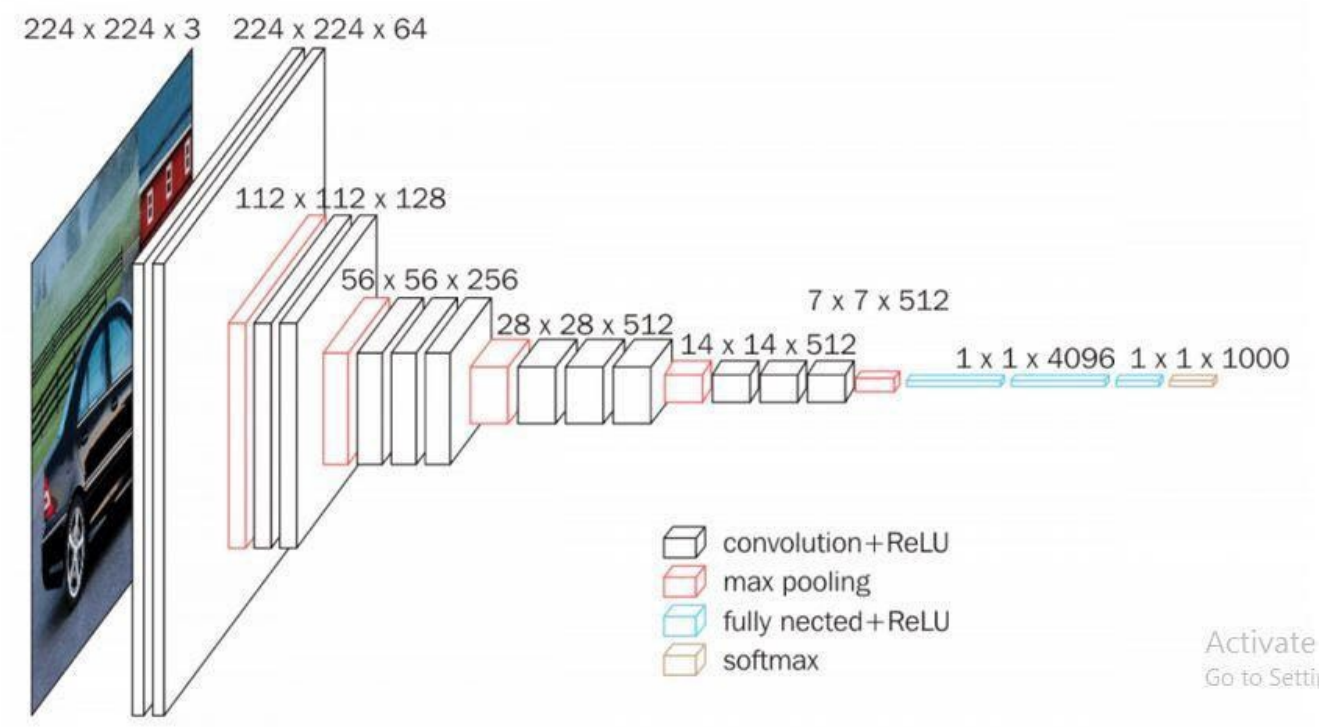
In [15]:
```
prepare = tf.keras.Sequential([
tf.keras.layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
tf.keras.layers.Rescaling(1./255)])
x_train_32 = prepare(X_train)
x_val_32 = prepare(x_val)
x_test_32 = prepare(x_val)
```

Vgg16

Architecture of VGG16 https://arxiv.org/pdf/1409.1556.pdf

- The 16 in VGG16 refers to 16 layers that have weights. In VGG16 there are thirteen convolutional layers, five Max Pooling layers, and three Dense layers which sum up to 21 layers but it has only sixteen weight layers i.e., learnable parameters layer.
- VGG16 takes input tensor size as 32,32 to 224, 244 with 3 RGB channel
- Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2.
- The convolution and max pool layers are consistently arranged throughout the whole architecture Conv-1 Layer has 64 number of filters, Conv-2 has 128 filters, Conv-3 has 256 filters, Conv 4 and Conv 5 has 512 filters.

- We will train only the last layer and two first layers ,other will be stay not trainable



In [31]:
```
vgg16=keras.applications.VGG16(include_top=False,weights='imagenet',input_shape=(IMAGE_SIZE,IMAGE_SIZE,3))
for layer in vgg16.layers:
    layer.trainable = False
for layer in vgg16.layers[-2:]:
    layer.trainable = True
for layer in vgg16.layers[1:3]:
    layer.trainable = True
vgg16.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 32, 32, 3)]       0

 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792

 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928

 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0

 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856

 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584

 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0

 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168

 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv3 (Conv2D)       (None, 8, 8, 256)         590080

 block3_pool (MaxPooling2D)  (None, 4, 4, 256)         0

 block4_conv1 (Conv2D)       (None, 4, 4, 512)         1180160

 block4_conv2 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 2, 2, 512)         0

 block5_conv1 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv2 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv3 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 2,398,528
Non-trainable params: 12,316,160
_________________________________________________________________
```

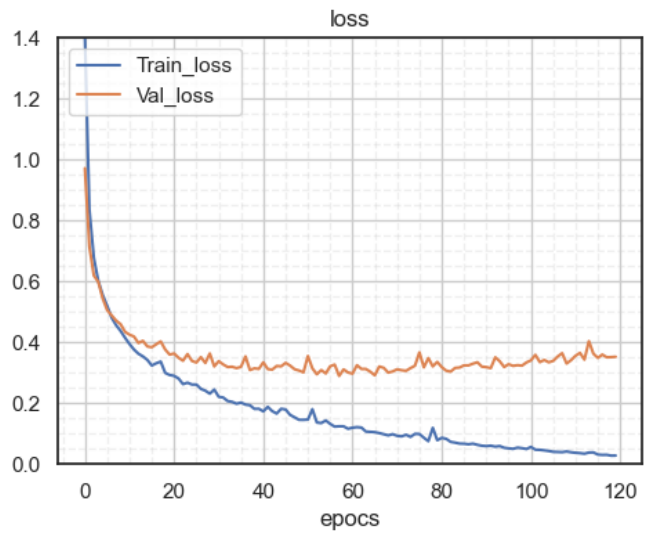
In [32]:
```
model2 = keras.Sequential()
model2.add(tf.keras.layers.InputLayer(input_shape=(IMAGE_SIZE,IMAGE_SIZE,3)))
model2.add(vgg16)
model2.add(layers.Flatten())
model2.add(layers.Dense(128,activation='relu'))
model2.add(layers.Dense(8, activation='softmax'))
model2.summary()
```

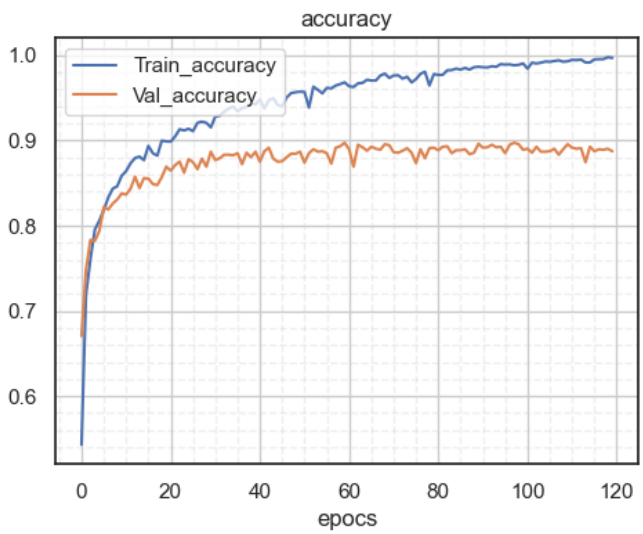
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 1, 1, 512)         14714688

 flatten_3 (Flatten)         (None, 512)               0

 dense_6 (Dense)             (None, 128)               65664

 dense_7 (Dense)             (None, 8)                 1032

=================================================================
Total params: 14,781,384
Trainable params: 2,465,224
Non-trainable params: 12,316,160
_________________________________________________________________
```

In [33]:
```
model2.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.00005),
              metrics=['accuracy'])
log_dir = "logs/CNN/"+'TL-VGG16-Flaten-FC128'
tb_callback = TensorBoard(log_dir=log_dir, histogram_freq=1,write_images=True)
```

In [ ]:
```
history=model2.fit(x_train_32, y_train, batch_size=64, epochs=120, validation_data=(x_val_32,y_val) ,callbacks = [tb_callbacl
```

In [ ]:
```
evaluation_plots(model2)
```





Its can be seen that a gap between losses and validation loss in-relation to previous option is decreased. Also accuracy slightly increased

Dropout

Dropout - is a regularization technique to prevent overfitting. It works by randomly "dropping out" a certain percentage of the nodes in the network during training. This means that the output of these nodes is set to zero and they are not included in the forward or backward pass. The effect of dropout is to effectively train an ensemble of models, with each model seeing a different subset of the nodes in the network.

In [25]:
```
prepare = tf.keras.Sequential([
tf.keras.layers.Rescaling(1./255)])
```

In [26]:
```
classifierD = keras.Sequential(name='CNN-128-MP-64-MP-D0.25-64-MP-Flaten-FC64')
prepare,
classifierD.add(layers.Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu',input_shape=(28,28,3)))
classifierD.add(layers.MaxPool2D(pool_size=(2,2)))
classifierD.add(layers.Conv2D(64, kernel_size=(3, 3),padding='same', activation='relu'))
classifierD.add(layers.MaxPool2D(pool_size=(2,2)))
classifierD.add(layers.Dropout(0.25))
classifierD.add(layers.Conv2D(64, kernel_size=(2, 2), activation='relu'))
classifierD.add(layers.MaxPool2D(pool_size=(2,2)))
classifierD.add(layers.Flatten())
classifierD.add(layers.Dense(64, activation='relu'))
```

```
classifierD.add(layers.Dense(8, activation='softmax'))
classifierD.summary()
```

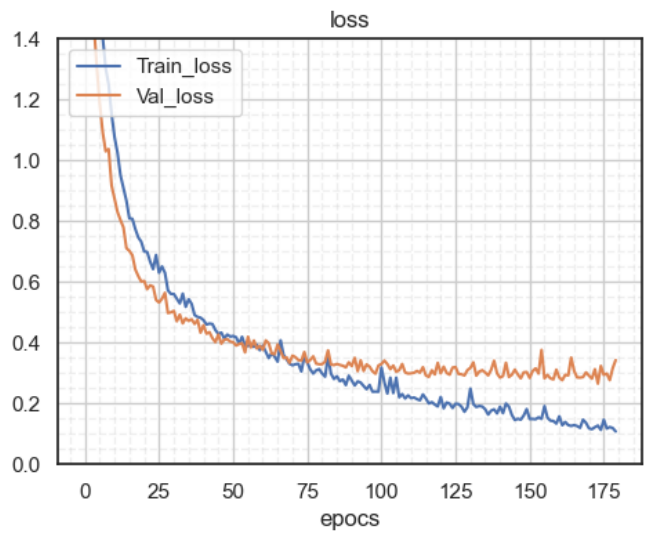
```
Model: "CNN-128-MP-64-MP-D0.25-64-MP-Flaten-FC64"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 28, 28, 128)       3584

 max_pooling2d_9 (MaxPooling  (None, 14, 14, 128)      0
 2D)

 conv2d_10 (Conv2D)          (None, 14, 14, 64)        73792

 max_pooling2d_10 (MaxPoolin  (None, 7, 7, 64)         0
 g2D)

 dropout (Dropout)           (None, 7, 7, 64)          0

 conv2d_11 (Conv2D)          (None, 6, 6, 64)          16448

 max_pooling2d_11 (MaxPoolin  (None, 3, 3, 64)         0
 g2D)

 flatten_3 (Flatten)         (None, 576)               0

 dense_6 (Dense)             (None, 64)                36928

 dense_7 (Dense)             (None, 8)                 520

=================================================================
Total params: 131,272
Trainable params: 131,272
Non-trainable params: 0
_________________________________________________________________
```

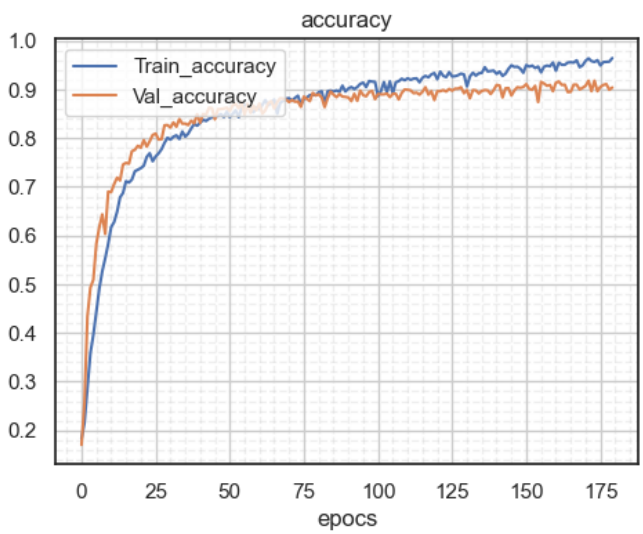
In [27]:
```
log_dir = "logs/CNN/"+'CNN-128-MP-64-MP-D0.25-64-MP-Flaten-FC64'
tb_callback = TensorBoard(log_dir=log_dir, histogram_freq=1,write_images=True)
```

In [28]:
```
classifierD.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.00005),
              metrics=['accuracy'])
```

In [ ]:
```
history=classifierD.fit(X_train, y_train, batch_size=64, epochs=180, validation_data=(x_val,y_val),callbacks=[tb_callback])
```

In [ ]:
```
evaluation_plots(classifierD)
```





It can be seen from the plot that a gap between training and validation losses are close to each other than previous options ,we prevented overefit.

Data Augmentation & Dropout

In [82]:
```
classifierDAd = keras.Sequential(name='CNN-Aug-128-MP-64-MP-D0.25-64-MP-Flaten-FC64')
data_augmentation,
classifierDAd.add(layers.Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu',input_shape=(28,28,3)))
classifierDAd.add(layers.MaxPool2D(pool_size=(2,2)))
classifierDAd.add(layers.Conv2D(64, kernel_size=(3, 3),padding='same', activation='relu'))
classifierDAd.add(layers.MaxPool2D(pool_size=(2,2)))
classifierDAd.add(layers.Dropout(0.25))
classifierDAd.add(layers.Conv2D(64, kernel_size=(2, 2), activation='relu'))
classifierDAd.add(layers.MaxPool2D(pool_size=(2,2)))
classifierDAd.add(layers.Flatten())
```

```
classifierDAd.add(layers.Dense(64, activation='relu'))
classifierDAd.add(layers.Dense(8, activation='softmax'))
classifierDAd.summary()
```

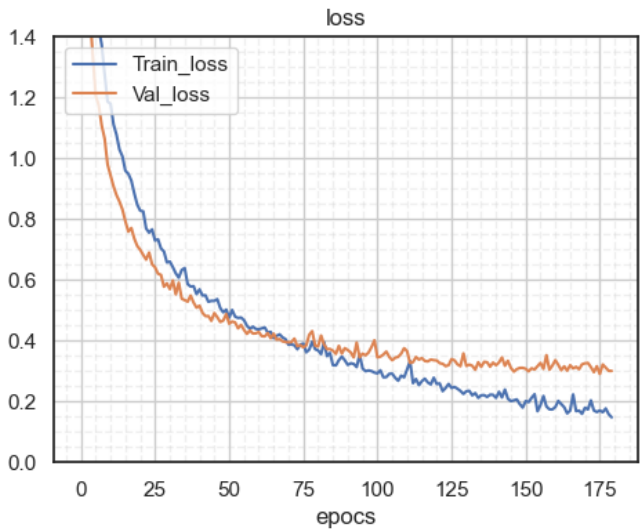
```
Model: "CNN-Aug-128-MP-64-MP-D0.25-64-MP-Flaten-FC64"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_27 (Conv2D)          (None, 28, 28, 128)       3584

 max_pooling2d_27 (MaxPoolin  (None, 14, 14, 128)      0
 g2D)

 conv2d_28 (Conv2D)          (None, 14, 14, 64)        73792

 max_pooling2d_28 (MaxPoolin  (None, 7, 7, 64)         0
 g2D)

 dropout (Dropout)           (None, 7, 7, 64)          0

 conv2d_29 (Conv2D)          (None, 6, 6, 64)          16448

 max_pooling2d_29 (MaxPoolin  (None, 3, 3, 64)         0
 g2D)

 flatten_9 (Flatten)         (None, 576)               0

 dense_18 (Dense)            (None, 64)                36928

 dense_19 (Dense)            (None, 8)                 520

=================================================================
Total params: 131,272
Trainable params: 131,272
Non-trainable params: 0
_________________________________________________________________
```

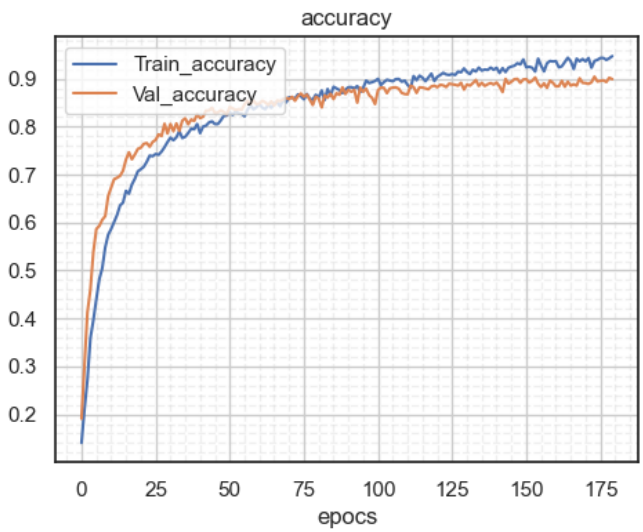
In [83]:
```
log_dir = "logs/CNN/"+'CNN-Aug-128-MP-64-MP-D0.25-64-MP-Flaten-FC64'
tb_callback = TensorBoard(log_dir=log_dir, histogram_freq=1,write_images=True)
```

In [84]:
```
classifierDAd.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.00005),
              metrics=['accuracy'])
```

In [ ]:
```
history=classifierDAd.fit(X_train, y_train, batch_size=64, epochs=180, validation_data=(x_val,y_val),callbacks=[tb_callback]
```

In [ ]:
```
evaluation_plots(classifierDAd)
```





Now ,we tried combine Data Augmentation technique and Dropout Layers to prevent overfiting ,it can be seen that a gap between training and validation losses are closer to each other than previous options. This combination improved previos scenario even more

L1 & L2

L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.

Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to quite an extent.

However, this regularization term differs in L1 and L2.

In L2, we have:

$$Loss = Error(Y - \widehat{Y}) + \lambda \sum_{1}^{n} w_i^2$$

- Here, lambda is the regularization parameter. It is the hyperparameter whose value is optimized for better results. L2 regularization is also known as weight decay as it forces the weights to decay towards zero.

In L1, we have:

$$Loss = Error(Y - \widehat{Y}) + \lambda \sum_{1}^{n} |w_i|$$

- In this, we penalize the absolute value of the weights.

In [31]:
```
prepare = tf.keras.Sequential([
tf.keras.layers.Rescaling(1./255)])
```

- Hyperparameter tuning refers to the process of choosing the optimal values for a machine learning model's hyperparameters. Hyperparameters are settings that determine the behavior of the model, and they are typically set prior to training the model. Hyperparameter tuning is an important step in the machine learning process because the performance of the model can be significantly influenced by the hyperparameters chosen.
- We will use this option to see the difference between values on L2 regularization

In [32]:
```
from tensorboard.plugins.hparams import api as hp
```

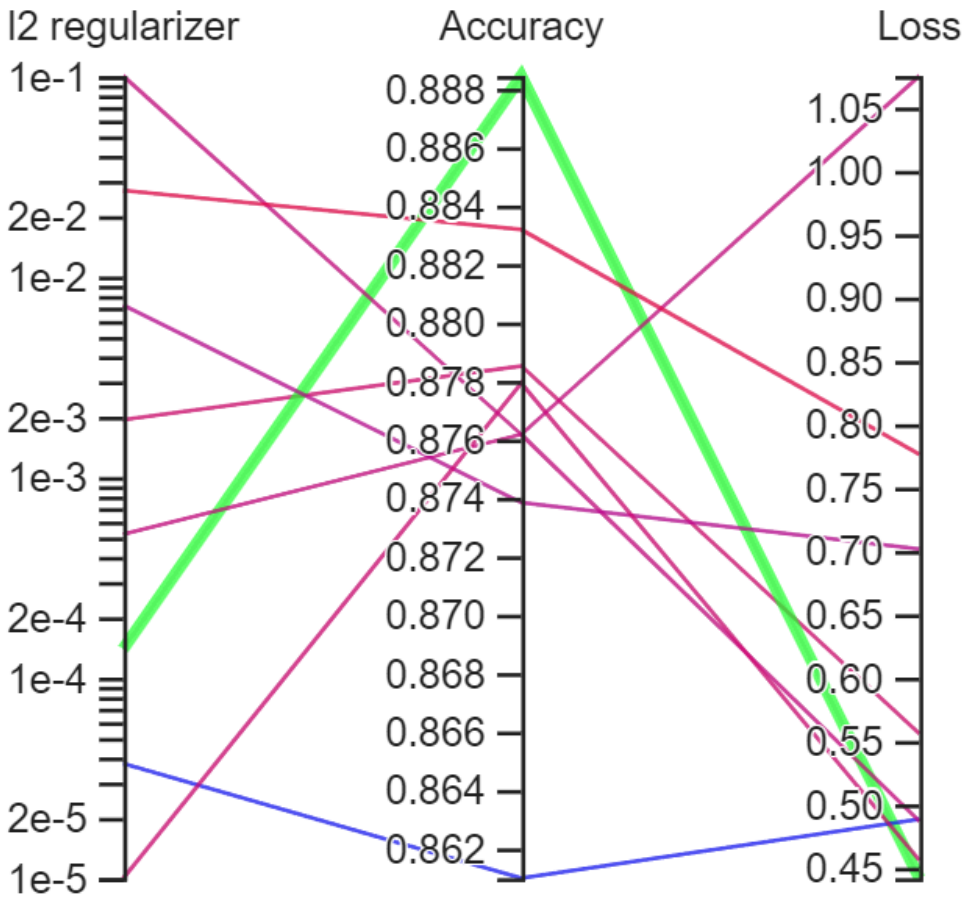
In [33]:
```
HP_L2 = hp.HParam('l2 regularizer', hp.RealInterval(1e-5,0.1))
METRIC_ACCURACY = 'accuracy'
METRIC_LOSS ='loss'
with tf.summary.create_file_writer('logs/hparam_tuning').as_default():
    hp.hparams_config(
       hparams=[HP_L2],
       metrics=[hp.Metric(METRIC_ACCURACY, display_name='Accuracy') ,hp.Metric(METRIC_LOSS, display_name='Loss')])
```

In [34]:
```
def train_test_model(hparams):
    classifier = keras.Sequential(name='CNN-128-MP-64-MP-64-MP-Flaten-FC64')
    prepare,
    classifier.add(layers.Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu',input_shape=(28,28,3)))
    classifier.add(layers.MaxPool2D(pool_size=(2,2)))
    classifier.add(layers.Conv2D(64, kernel_size=(3, 3),padding='same', activation='relu'))
    classifier.add(layers.MaxPool2D(pool_size=(2,2)))
    classifier.add(layers.Conv2D(64, kernel_size=(2, 2), activation='relu'))
    classifier.add(layers.MaxPool2D(pool_size=(2,2)))
    classifier.add(layers.Flatten())
    classifier.add(layers.Dense(64, activation='relu' ,kernel_regularizer=keras.regularizers.L2(l2=hparams[HP_L2])))
    classifier.add(layers.Dense(8, activation='softmax'))


    classifier.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.00005),
              metrics=['accuracy'])

    classifier.fit(X_train, y_train, batch_size=64, epochs=180, validation_data=(x_val,y_val))
    loss, accuracy = classifier.evaluate(x_val,y_val)
    return (loss ,accuracy)
```

In [35]:
```
def run(run_dir, hparams):
    with tf.summary.create_file_writer(run_dir).as_default():
        hp.hparams(hparams)  # record the values used in this trial
        loss,accuracy = train_test_model(hparams)
        tf.summary.scalar(METRIC_LOSS,loss, step=1)
        tf.summary.scalar(METRIC_ACCURACY, accuracy, step=1)
```

In [ ]:
```
session_num = 0
for l2 in np.logspace(np.log10(HP_L2.domain.min_value), np.log10(HP_L2.domain.max_value),8):
    hparams = {HP_L2: l2}
    run_name = "run-%d" % session_num
    print('--- Starting trial: %s' % run_name)
    print({h.name: hparams[h] for h in hparams})
    run('logs/hparam_tuning/' + run_name, hparams)
    session_num += 1
```

In [ ]:
```
%reload_ext tensorboard
%tensorboard --logdir logs/hparam_tuning
```



From the image above can be seen that for diffirent l2 factors ve get different accuracy and loss

- Now lets run couple of difference L2 regularization factors

In [45]:
```
classifier = keras.Sequential(name='CNN-128-MP-64-MP-64-MP-Flaten-FC64_L2')
prepare,
classifier.add(layers.Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu',input_shape=(28,28,3)))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(3, 3),padding='same', activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(2, 2), activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Flatten())
classifier.add(layers.Dense(64, activation='relu' ,kernel_regularizer=keras.regularizers.L2(l2=1e-2)))
classifier.add(layers.Dense(8, activation='softmax'))
classifier.summary()
```

```
Model: "CNN-128-MP-64-MP-64-MP-Flaten-FC64_L2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_39 (Conv2D)          (None, 28, 28, 128)       3584

 max_pooling2d_39 (MaxPoolin  (None, 14, 14, 128)      0
 g2D)

 conv2d_40 (Conv2D)          (None, 14, 14, 64)        73792

 max_pooling2d_40 (MaxPoolin  (None, 7, 7, 64)         0
 g2D)

 conv2d_41 (Conv2D)          (None, 6, 6, 64)          16448

 max_pooling2d_41 (MaxPoolin  (None, 3, 3, 64)         0
 g2D)

 flatten_13 (Flatten)        (None, 576)               0

 dense_26 (Dense)            (None, 64)                36928

 dense_27 (Dense)            (None, 8)                 520

=================================================================
Total params: 131,272
Trainable params: 131,272
Non-trainable params: 0
_________________________________________________________________
```

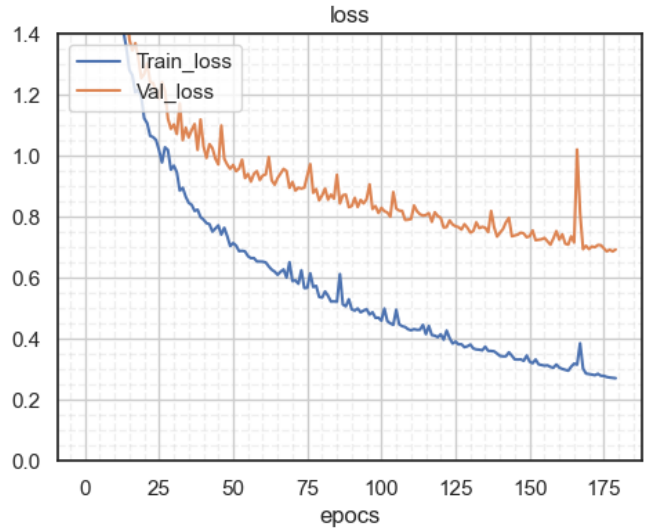
In [46]:
```
classifier.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.00005),
              metrics=['accuracy'])
```

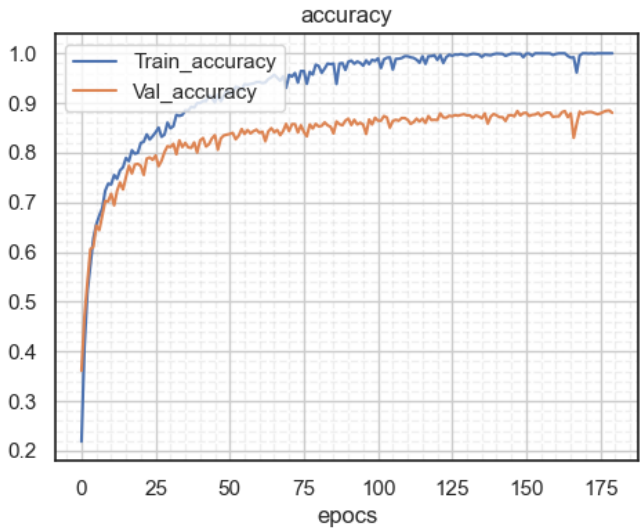
In [47]:
```
log_dir = "logs/CNN/"+'CNN-128-MP-64-MP-64-MP-Flaten-FC64_L2'
tb_callback = TensorBoard(log_dir=log_dir, histogram_freq=1,write_images=True)
```

In [ ]:
```
classifier.fit(X_train, y_train, batch_size=64, epochs=180, validation_data=(x_val,y_val),callbacks=[tb_callback])
```

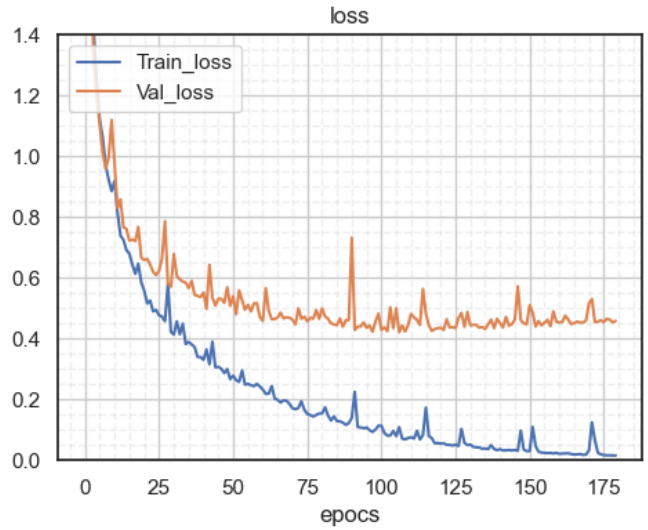
In [ ]:
```
evaluation_plots(classifier)
```

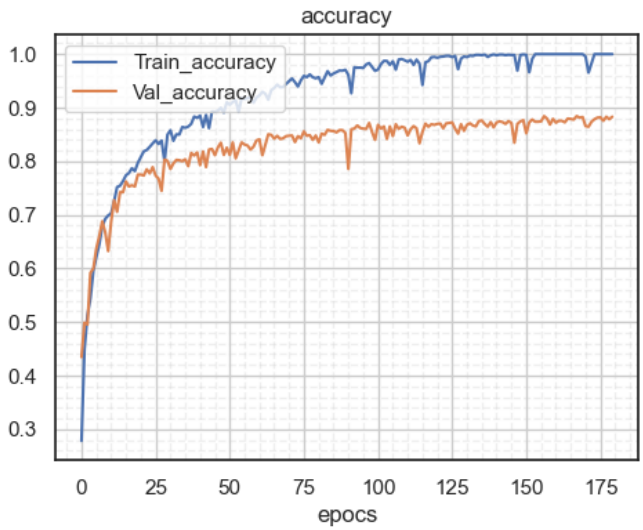
L2 = 10^-2





L2 =10^-4





Early stoping

Early stopping is a technique used to prevent overfitting in machine learning. It works by monitoring the performance of a model on a validation set during training. If the model's performance on the validation set starts to degrade, training is stopped. This can be particularly useful when training deep learning models, as they can have a tendency to overfit to the training data if they are trained for too long.

In the above image, we will stop training at the dotted line since after that our model will start overfitting on the training data

In [50]:
```
prepare = tf.keras.Sequential([
tf.keras.layers.Rescaling(1./255)])
```

In [51]:
```
classifier = keras.Sequential(name='CNN-128-MP-64-MP-64-MP-Flaten-FC64-ES')
prepare,
classifier.add(layers.Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu',input_shape=(28,28,3)))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(3, 3),padding='same', activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Conv2D(64, kernel_size=(2, 2), activation='relu'))
classifier.add(layers.MaxPool2D(pool_size=(2,2)))
classifier.add(layers.Flatten())
classifier.add(layers.Dense(64, activation='relu'))
classifier.add(layers.Dense(8, activation='softmax'))
classifier.summary()
```

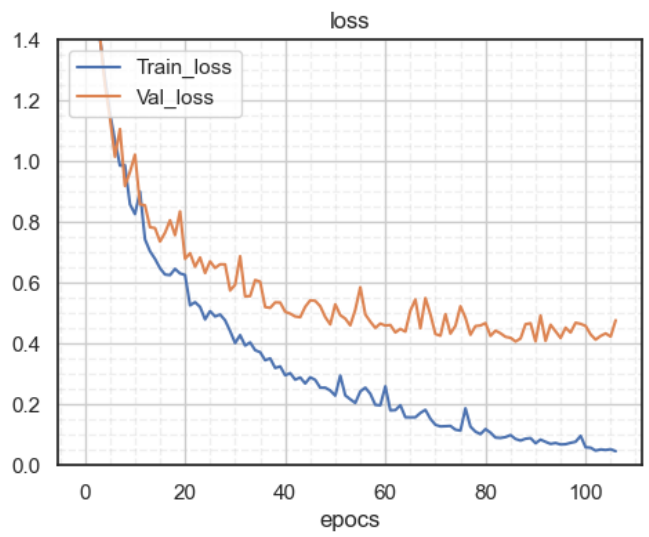
```
Model: "CNN-128-MP-64-MP-64-MP-Flaten-FC64-ES"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_42 (Conv2D)          (None, 28, 28, 128)       3584

 max_pooling2d_42 (MaxPoolin  (None, 14, 14, 128)      0
 g2D)

 conv2d_43 (Conv2D)          (None, 14, 14, 64)        73792

 max_pooling2d_43 (MaxPoolin  (None, 7, 7, 64)         0
 g2D)

 conv2d_44 (Conv2D)          (None, 6, 6, 64)          16448

 max_pooling2d_44 (MaxPoolin  (None, 3, 3, 64)         0
 g2D)

 flatten_14 (Flatten)        (None, 576)               0

 dense_28 (Dense)            (None, 64)                36928

 dense_29 (Dense)            (None, 8)                 520

=================================================================
Total params: 131,272
Trainable params: 131,272
Non-trainable params: 0
_________________________________________________________________
```

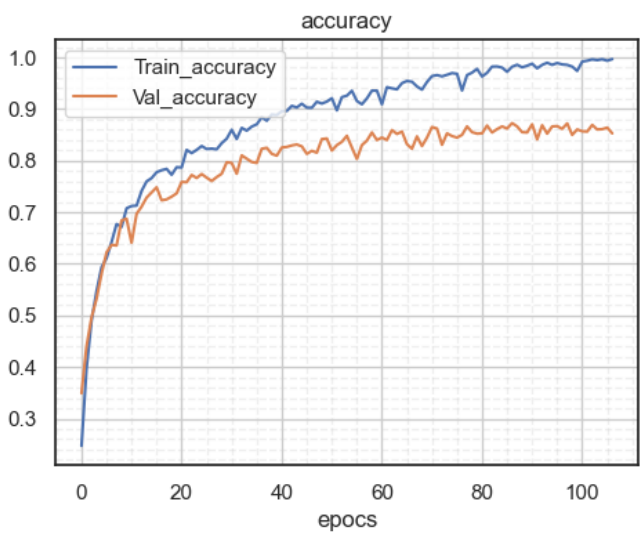
In [52]:
```
log_dir = "logs/CNN/"+'CNN-128-MP-64-MP-64-MP-Flaten-FC64-ES'
tb_callback = TensorBoard(log_dir=log_dir, histogram_freq=1,write_images=True)
es_callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=20)
```

In [53]:
```
classifier.compile(loss='sparse_categorical_crossentropy',
              optimizer=keras.optimizers.Adam(learning_rate=0.00005),
              metrics=['accuracy'])
```

In [ ]:
```
history=classifier.fit(X_train, y_train, batch_size=64, epochs=180, validation_data=(x_val,y_val),callbacks=[tb_callback ,es_
```

In [ ]:
```
evaluation_plots(classifier)
```





Here we used early stopping technique that involves interrupting training when the model's performance without improvments during 20 epocs ,as we see we runed 180 epocs ,but training was interupted after 107 epocs.

Combined Model

Transfer Learning + Data Augmentation + DropOuts+ EarlyStoping

In [16]:
```
IMAGE_SIZE =32
```

In [17]:
```
prepare = tf.keras.Sequential([
tf.keras.layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
tf.keras.layers.Rescaling(1./255)])
x_train_32 = prepare(X_train)
x_val_32 = prepare(x_val)
x_test_32 = prepare(x_val)
```

In [18]:
```
data_augmentation = tf.keras.Sequential([
  layers.RandomRotation(0.4),
  layers.RandomFlip(),
  layers.RandomTranslation(0.15,0.15),
  layers.RandomContrast(0.3),
  layers.RandomBrightness(0.3,value_range=(0, 1)),
])
```

In [19]:
```
vgg19=keras.applications.VGG19(include_top=False,weights='imagenet',input_shape=(IMAGE_SIZE,IMAGE_SIZE,3))
for layer in vgg19.layers:
    layer.trainable = False
for layer in vgg19.layers[-2:]:
    layer.trainable = True
for layer in vgg19.layers[1:3]:
    layer.trainable = True
vgg19.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792

 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928

 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0

 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856

 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584

 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0

 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168

 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv3 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv4 (Conv2D)       (None, 8, 8, 256)         590080

 block3_pool (MaxPooling2D)  (None, 4, 4, 256)         0

 block4_conv1 (Conv2D)       (None, 4, 4, 512)         1180160

 block4_conv2 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_conv4 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 2, 2, 512)         0

 block5_conv1 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv2 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv3 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv4 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 2,398,528
Non-trainable params: 17,625,856
_________________________________________________________________
```

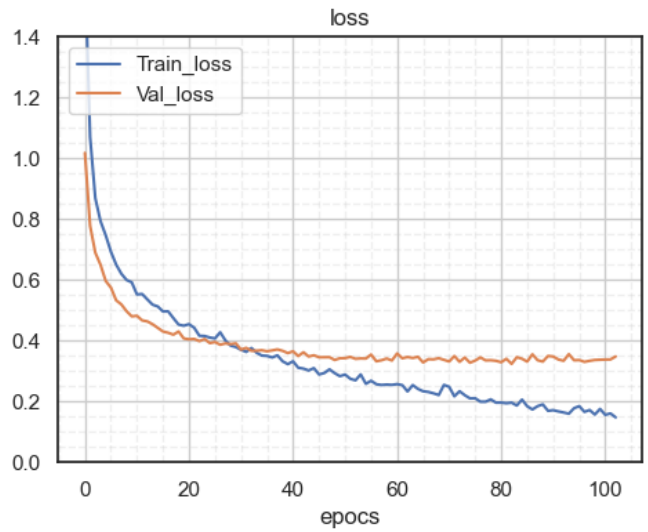
In [20]:
```
model4 = keras.Sequential()
data_augmentation,
model4.add(tf.keras.layers.InputLayer(input_shape=(IMAGE_SIZE,IMAGE_SIZE,3)))
model4.add(vgg19)
model4.add(layers.Dropout(0.3))
model4.add(layers.Flatten())
model4.add(layers.Dense(128,activation='relu'))
model4.add(layers.Dropout(0.3))
model4.add(layers.Dense(8, activation='softmax'))
model4.summary()
```

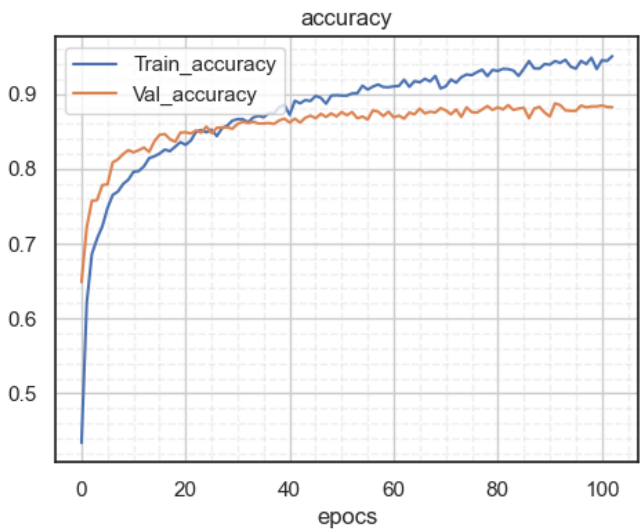
```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384

 dropout (Dropout)           (None, 1, 1, 512)         0

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 128)               65664

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 8)                 1032

=================================================================
Total params: 20,091,080
Trainable params: 2,465,224
Non-trainable params: 17,625,856
_________________________________________________________________
```

In [21]:
```
model4.compile(loss='sparse_categorical_crossentropy',
               optimizer=keras.optimizers.Adam(learning_rate=0.00005),
               metrics=['accuracy'])
log_dir = "logs/CNN/"+'TL-D0.3-VGG19-Flaten-FC128-D0.3+ES'
tb_callback = TensorBoard(log_dir=log_dir, histogram_freq=1,write_images=True)
es_callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=20)
```

In [ ]:
```
history=model4.fit(x_train_32, y_train, batch_size=64, epochs=140, validation_data=(x_val_32,y_val) ,callbacks = [es_callbacl
```

In [ ]:
```
evaluation_plots(model4)
```





Now we used combination with diffirent options to prevent an overfiting

In [27]:
```
%load_ext tensorboard
%tensorboard --logdir logs/CNN
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```