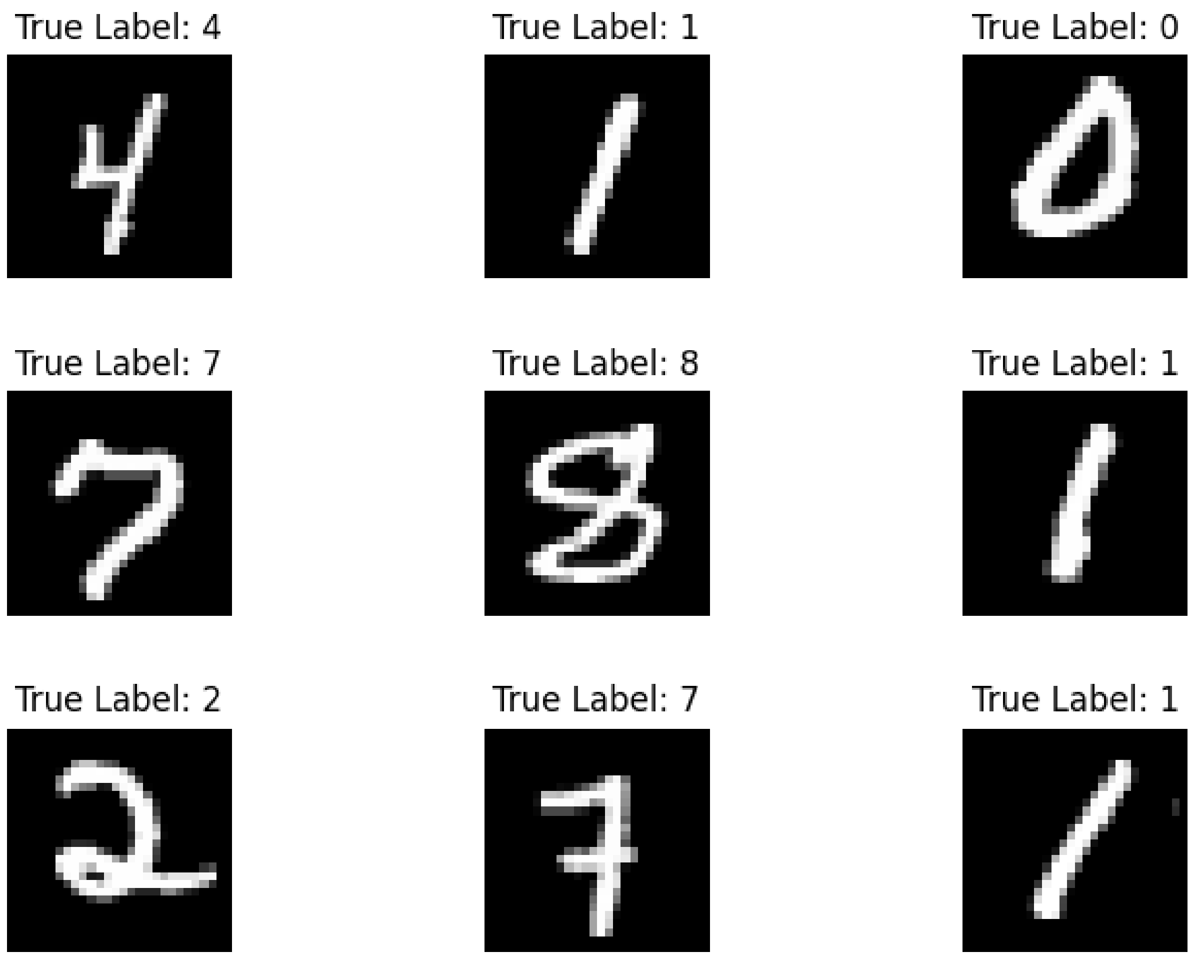
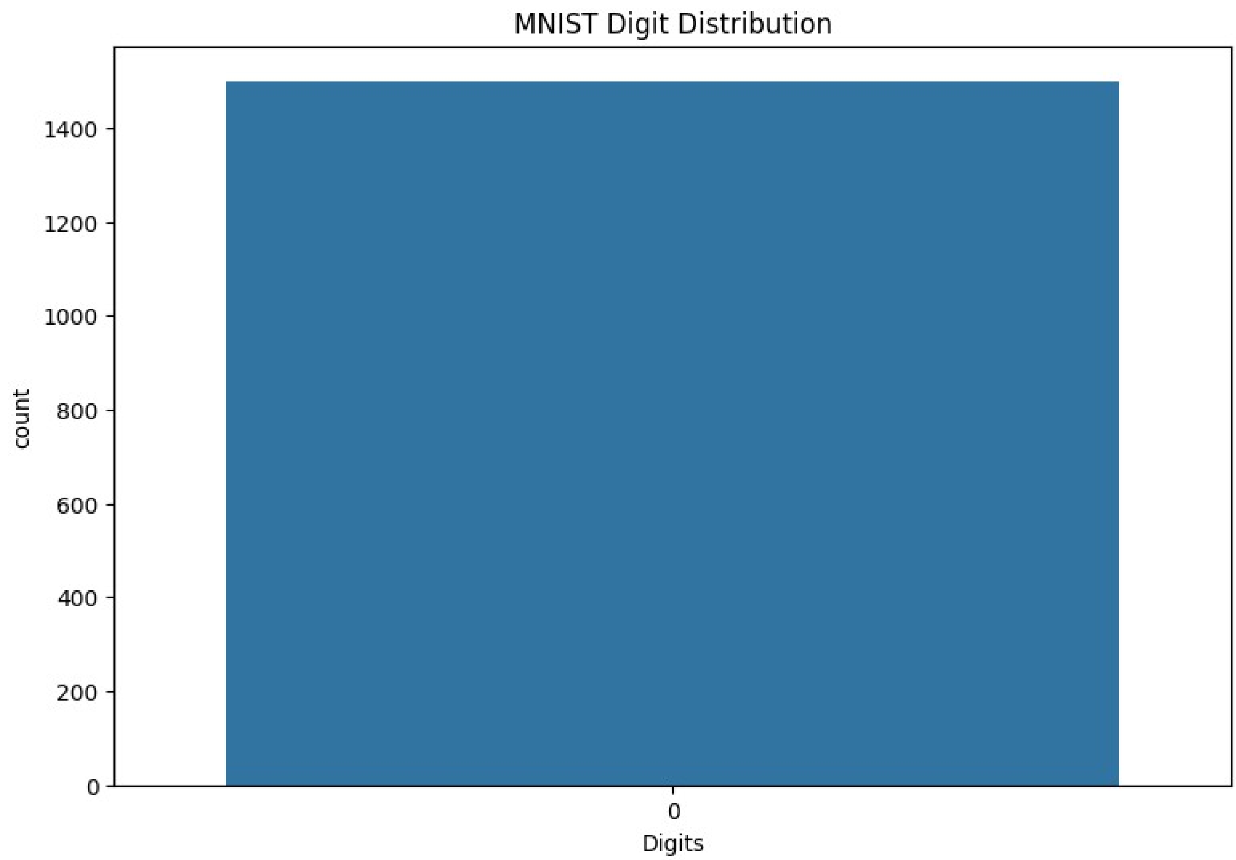
```
ion_minor":0}
```

```
Dataset mnist downloaded and prepared to
/root/tensorflow_datasets/mnist/3.0.1. Subsequent calls will reuse
this data.
```

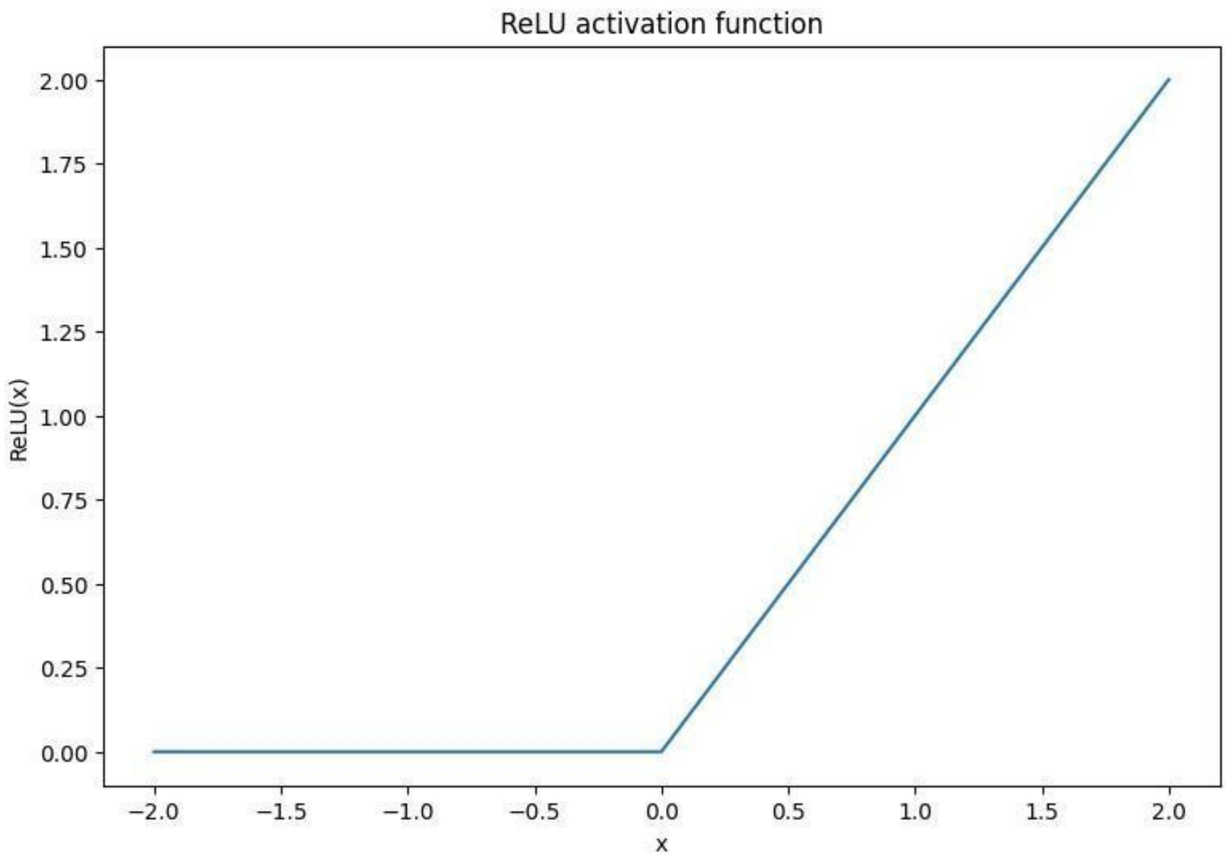
```
x_viz, y_viz = tfds.load("mnist", split=['train[:1500]'], batch_size=-
1, as_supervised=True)[0]
x_viz = tf.squeeze(x_viz, axis=3)

for i in range(9):
    plt.subplot(3,3,1+i)
    plt.axis('off')
    plt.imshow(x_viz[i], cmap='gray')
    plt.title(f"True Label: {y_viz[i]}")
    plt.subplots_adjust(hspace=.5)
```



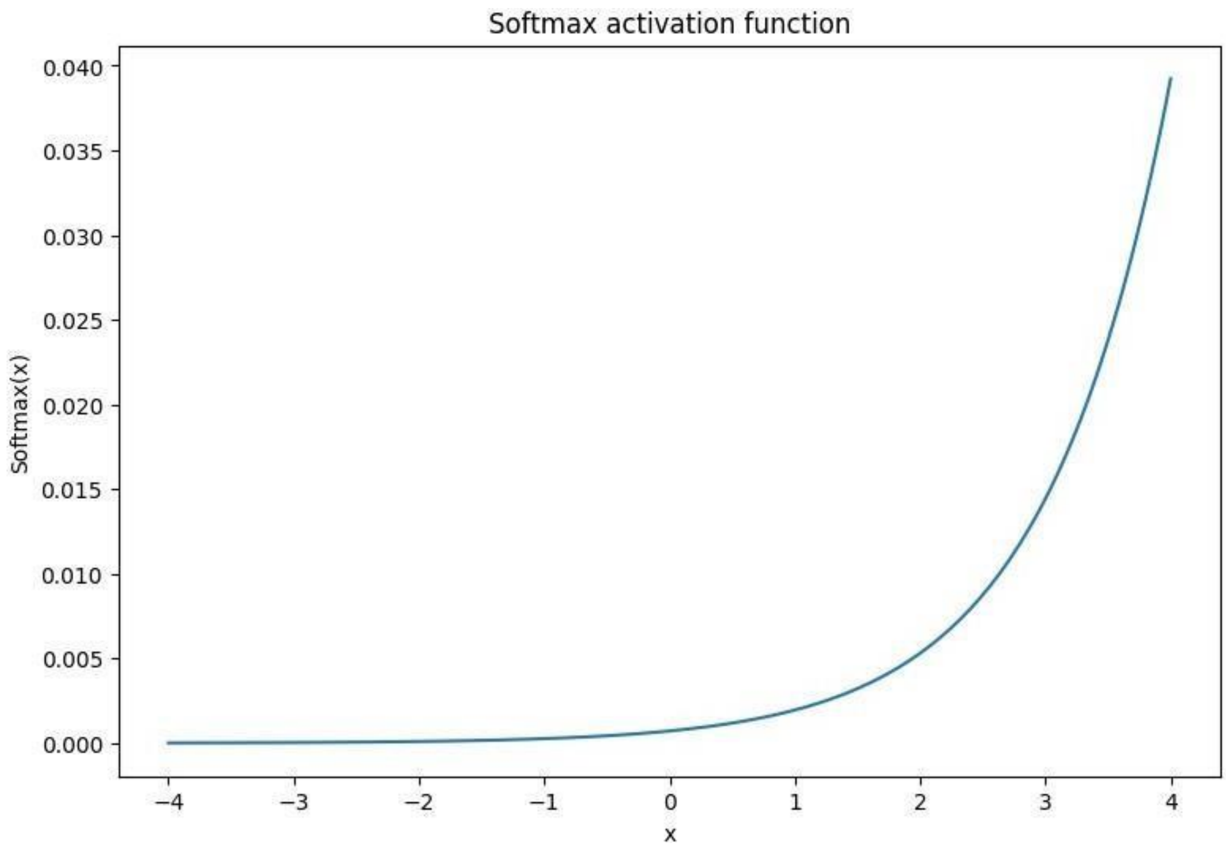

```
sns.countplot(y_viz.numpy());
plt.xlabel('Digits')
plt.title("MNIST Digit Distribution");
```




```
def preprocess(x, y):
  # Reshaping the data
  x = tf.reshape(x, shape=[-1, 784])
  # Rescaling the data
  x = x/255
  return x, y

train_data, val_data = train_data.map(preprocess),
val_data.map(preprocess)

x = tf.linspace(-2, 2, 201)
x = tf.cast(x, tf.float32)
plt.plot(x, tf.nn.relu(x));
plt.xlabel('x')
plt.ylabel('ReLU(x)')
plt.title('ReLU activation function');
```




```
x = tf.linspace(-4, 4, 201)
x = tf.cast(x, tf.float32)
plt.plot(x, tf.nn.softmax(x, axis=0));
plt.xlabel('x')
plt.ylabel('Softmax(x)')
plt.title('Softmax activation function');
```




```
def xavier_init(shape):
  # Computes the xavier initialization values for a weight matrix
  in_dim, out_dim = shape
  xavier_lim = tf.sqrt(6.)/tf.sqrt(tf.cast(in_dim + out_dim,
tf.float32))
  weight_vals = tf.random.uniform(shape=(in_dim, out_dim),
                                  minval=-xavier_lim,
maxval=xavier_lim, seed=22)
  return weight_vals

class DenseLayer(tf.Module):

  def_init_(self, out_dim, weight_init=xavier_init,
activation=tf.identity):
    # Initialize the dimensions and activation functions
    self.out_dim = out_dim
    self.weight_init = weight_init
    self.activation = activation
    self.built = False

  def__call__(self, x):
    if not self.built:
      # Infer the input dimension based on first call
      self.in_dim = x.shape[1]
```

```python
      # Initialize the weights and biases using Xavier scheme
      self.w = tf.Variable(xavier_init(shape=(self.in_dim,
self.out_dim)))
      self.b = tf.Variable(tf.zeros(shape=(self.out_dim,)))
      self.built = True
    # Compute the forward pass
    z = tf.add(tf.matmul(x, self.w), self.b)
    return self.activation(z)

class MLP(tf.Module):

  def_init_(self, layers):
    self.layers = layers

  @tf.function
  def___call__(self, x, preds=False):
    # Execute the model's layers sequentially
    for layer in self.layers:
      x = layer(x)
    return x

hidden_layer_1_size = 700
hidden_layer_2_size = 500
output_size = 10

mlp_model = MLP([
    DenseLayer(out_dim=hidden_layer_1_size, activation=tf.nn.relu),
    DenseLayer(out_dim=hidden_layer_2_size, activation=tf.nn.relu),
    DenseLayer(out_dim=output_size)])

def cross_entropy_loss(y_pred, y):
  # Compute cross entropy loss with a sparse operation
  sparse_ce = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,
logits=y_pred)
  return tf.reduce_mean(sparse_ce)

def accuracy(y_pred, y):
  # Compute accuracy after extracting class predictions
  class_preds = tf.argmax(tf.nn.softmax(y_pred), axis=1)
  is_equal = tf.equal(y, class_preds)
  return tf.reduce_mean(tf.cast(is_equal, tf.float32))

class Adam:

    def_init_(self, learning_rate=1e-3, beta_1=0.9, beta_2=0.999,
ep=1e-7):
      # Initialize optimizer parameters and variable slots
      self.beta_1 = beta_1
      self.beta_2 = beta_2
      self.learning_rate = learning_rate
      self.ep = ep
```

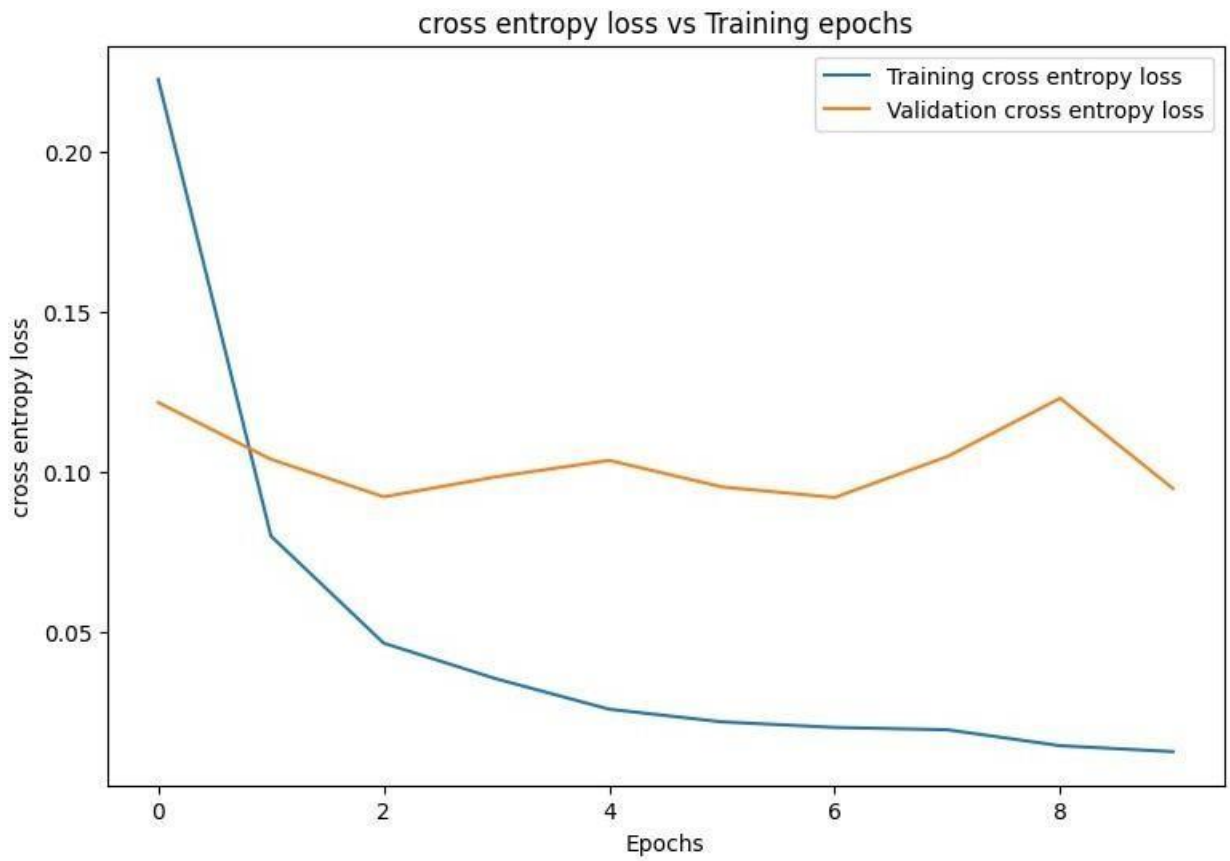
```
      self.t = 1.
      self.v_dvar, self.s_dvar = [], []
      self.built = False

    def apply_gradients(self, grads, vars):
      # Initialize variables on the first call
      if not self.built:
        for var in vars:
          v = tf.Variable(tf.zeros(shape=var.shape))
          s = tf.Variable(tf.zeros(shape=var.shape))
          self.v_dvar.append(v)
          self.s_dvar.append(s)
        self.built = True
      # Update the model variables given their gradients
      for i, (d_var, var) in enumerate(zip(grads, vars)):
        self.v_dvar[i].assign(self.beta_1*self.v_dvar[i] + (1-
self.beta_1)*d_var)
        self.s_dvar[i].assign(self.beta_2*self.s_dvar[i] + (1-
self.beta_2)*tf.square(d_var))
        v_dvar_bc = self.v_dvar[i]/(1-(self.beta_1**self.t))
        s_dvar_bc = self.s_dvar[i]/(1-(self.beta_2**self.t))

var.assign_sub(self.learning_rate*(v_dvar_bc/(tf.sqrt(s_dvar_bc) +
self.ep)))
      self.t += 1.
      return

def train_step(x_batch, y_batch, loss, acc, model, optimizer):
  # Update the model state given a batch of data
  with tf.GradientTape() as tape:
    y_pred = model(x_batch)
    batch_loss = loss(y_pred, y_batch)
  batch_acc = acc(y_pred, y_batch)
  grads = tape.gradient(batch_loss, model.variables)
  optimizer.apply_gradients(grads, model.variables)
  return batch_loss, batch_acc

def val_step(x_batch, y_batch, loss, acc, model):
  # Evaluate the model on given a batch of validation data
  y_pred  =  model(x_batch)
  batch_loss = loss(y_pred, y_batch)
  batch_acc = acc(y_pred, y_batch)
  return batch_loss, batch_acc

def train_model(mlp, train_data, val_data, loss, acc, optimizer,
epochs):
  # Initialize data structures
  train_losses, train_accs = [], []
  val_losses, val_accs = [], []
```

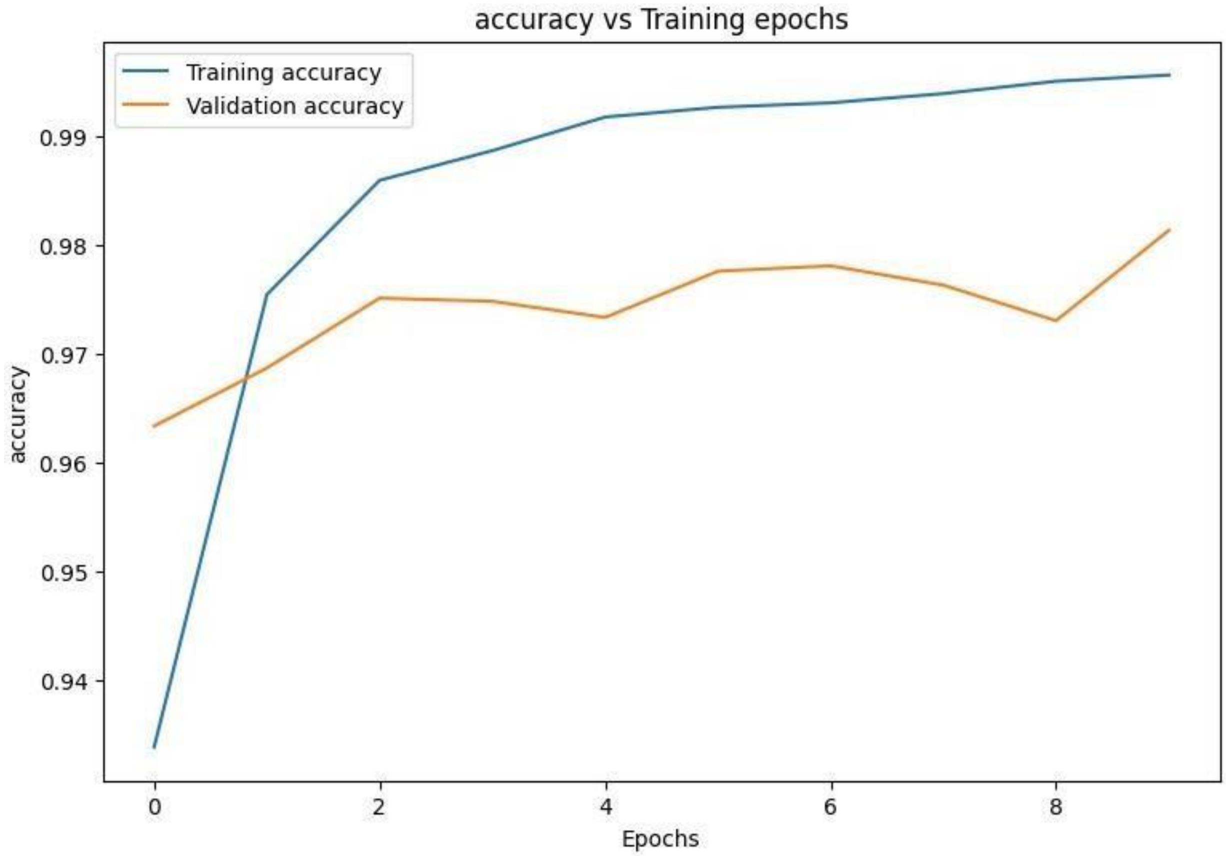
```
  # Format training loop and begin training
  for epoch in range(epochs):
    batch_losses_train, batch_accs_train = [], []
    batch_losses_val, batch_accs_val = [], []

    # Iterate over the training data
    for x_batch, y_batch in train_data:
      # Compute gradients and update the model's parameters
      batch_loss, batch_acc = train_step(x_batch, y_batch, loss, acc,
mlp, optimizer)
      # Keep track of batch-level training performance
      batch_losses_train.append(batch_loss)
      batch_accs_train.append(batch_acc)

    # Iterate over the validation data
    for x_batch, y_batch in val_data:
      batch_loss, batch_acc = val_step(x_batch, y_batch, loss, acc,
mlp)
      batch_losses_val.append(batch_loss)
      batch_accs_val.append(batch_acc)

    # Keep track of epoch-level model performance
    train_loss, train_acc = tf.reduce_mean(batch_losses_train),
tf.reduce_mean(batch_accs_train)
    val_loss, val_acc = tf.reduce_mean(batch_losses_val),
tf.reduce_mean(batch_accs_val)
    train_losses.append(train_loss)
    train_accs.append(train_acc)
    val_losses.append(val_loss)
    val_accs.append(val_acc)
    print(f"Epoch: {epoch}")
    print(f"Training loss: {train_loss:.3f}, Training accuracy:
{train_acc:.3f}")
    print(f"Validation loss: {val_loss:.3f}, Validation accuracy:
{val_acc:.3f}")
  return train_losses, train_accs, val_losses, val_accs

train_losses, train_accs, val_losses, val_accs =
train_model(mlp_model, train_data, val_data,

loss=cross_entropy_loss, acc=accuracy,

optimizer=Adam(), epochs=10)
```

```
Epoch: 0
Training loss: 0.223, Training accuracy: 0.934
Validation loss: 0.122, Validation accuracy: 0.963
Epoch: 1
Training loss: 0.080, Training accuracy: 0.975
Validation loss: 0.104, Validation accuracy: 0.969
```

```
Epoch: 2
Training loss: 0.046, Training accuracy: 0.986
Validation loss: 0.092, Validation accuracy: 0.975
Epoch: 3
Training loss: 0.035, Training accuracy: 0.989
Validation loss: 0.098, Validation accuracy: 0.975
Epoch: 4
Training loss: 0.026, Training accuracy: 0.992
Validation loss: 0.104, Validation accuracy: 0.973
Epoch: 5
Training loss: 0.022, Training accuracy: 0.993
Validation loss: 0.095, Validation accuracy: 0.978
Epoch: 6
Training loss: 0.020, Training accuracy: 0.993
Validation loss: 0.092, Validation accuracy: 0.978
Epoch: 7
Training loss: 0.019, Training accuracy: 0.994
Validation loss: 0.105, Validation accuracy: 0.976
Epoch: 8
Training loss: 0.014, Training accuracy: 0.995
Validation loss: 0.123, Validation accuracy: 0.973
Epoch: 9
Training loss: 0.012, Training accuracy: 0.996
Validation loss: 0.095, Validation accuracy: 0.981
```

```
def plot_metrics(train_metric, val_metric, metric_type):
  # Visualize metrics vs training Epochs
  plt.figure()
  plt.plot(range(len(train_metric)), train_metric, label = f"Training
{metric_type}")
  plt.plot(range(len(val_metric)), val_metric, label = f"Validation
{metric_type}")
  plt.xlabel("Epochs")
  plt.ylabel(metric_type)
  plt.legend()
  plt.title(f"{metric_type} vs Training epochs");
plot_metrics(train_losses, val_losses, "cross entropy loss")
plot_metrics(train_accs, val_accs, "accuracy")
```


cross entropy loss vs Training epochs
Training cross entropy loss
Validation cross entropy loss
cross entropy loss
0.20
0.15
0.10
0.05
0
2
4
6
8
Epochs


accuracy vs Training epochs
Training accuracy
Validation accuracy
0.99
0.98
0.97
0.96
0.95
0.94
accuracy
0
2
4
6
8
Epochs

In [1]:

```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

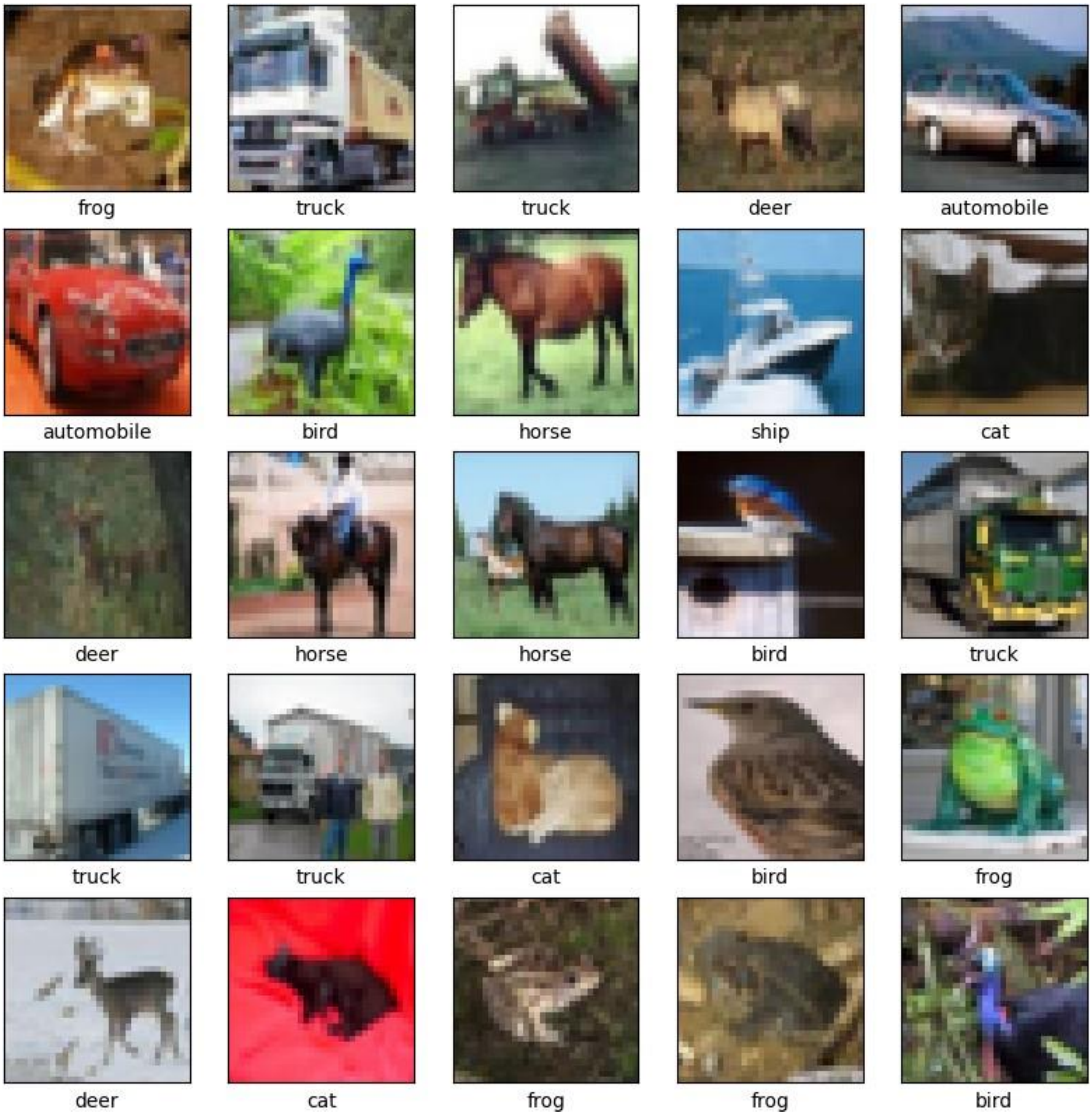
In [2]:

```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.ta
r.gz (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)
170498071/170498071 [==============================] - 4s 0us/step
```

In [3]:

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```



In [4]:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

In [5]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
_________________________________________________________________
```

In [6]:

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

In [7]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928     
                                                                 
 flatten (Flatten)           (None, 1024)              0         
                                                                 
 dense (Dense)               (None, 64)                65600     
                                                                 
 dense_1 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

In [8]:

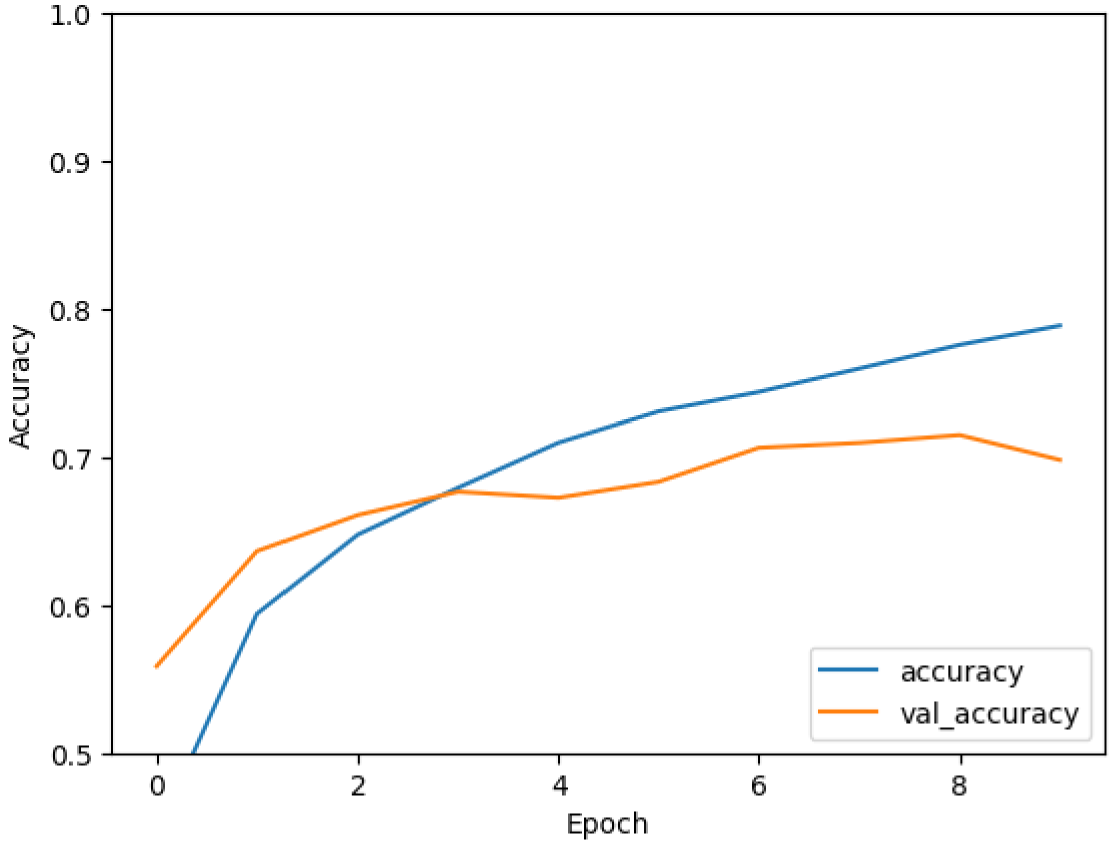
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 98s 62ms/step - loss: 1.5136
- accuracy: 0.4456 - val_loss: 1.2242 - val_accuracy: 0.5587
Epoch 2/10
1563/1563 [==============================] - 87s 56ms/step - loss: 1.1542
- accuracy: 0.5941 - val_loss: 1.0390 - val_accuracy: 0.6365
Epoch 3/10
1563/1563 [==============================] - 84s 54ms/step - loss: 1.0065
- accuracy: 0.6477 - val_loss: 0.9688 - val_accuracy: 0.6609
Epoch 4/10
1563/1563 [==============================] - 86s 55ms/step - loss: 0.9125
- accuracy: 0.6794 - val_loss: 0.9298 - val_accuracy: 0.6767
Epoch 5/10
1563/1563 [==============================] - 88s 57ms/step - loss: 0.8304
- accuracy: 0.7098 - val_loss: 0.9642 - val_accuracy: 0.6726
Epoch 6/10
1563/1563 [==============================] - 86s 55ms/step - loss: 0.7719
- accuracy: 0.7309 - val_loss: 0.9101 - val_accuracy: 0.6834
Epoch 7/10
1563/1563 [==============================] - 85s 55ms/step - loss: 0.7254
- accuracy: 0.7439 - val_loss: 0.8851 - val_accuracy: 0.7064
Epoch 8/10
1563/1563 [==============================] - 85s 55ms/step - loss: 0.6837
- accuracy: 0.7597 - val_loss: 0.8545 - val_accuracy: 0.7097
Epoch 9/10
1563/1563 [==============================] - 86s 55ms/step - loss: 0.6376
- accuracy: 0.7757 - val_loss: 0.8599 - val_accuracy: 0.7150
Epoch 10/10
1563/1563 [==============================] - 90s 58ms/step - loss: 0.6030
- accuracy: 0.7887 - val_loss: 0.9074 - val_accuracy: 0.6982
```

In [9]:

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 6s - loss: 0.9074 - accuracy: 0.6982 - 6s/epoch - 19ms/step
```



In [10]:

```
print(test_acc)
```

```
0.698199987411499
```