SOURCE CODES

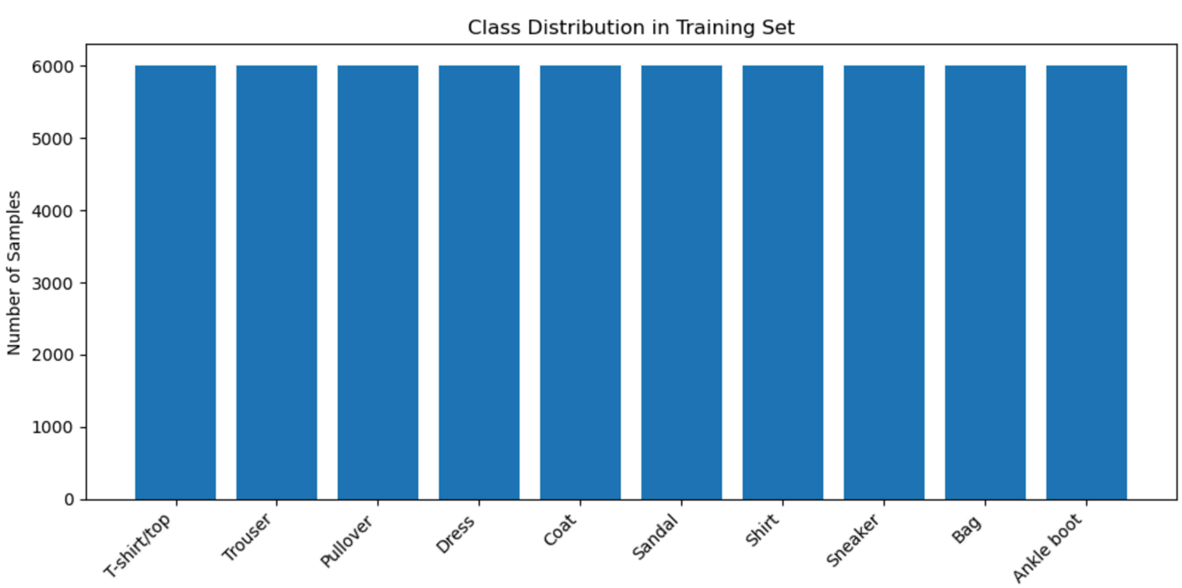
[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import fashion_mnist

# Load the Fashion MNIST dataset
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

[2]:
```python
print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
print("Image size:", X_train[0].shape)
print("Data type:", X_train.dtype)
print("Pixel value range:", X_train.min(), "to", X_train.max())
```

```
Training set shape: (60000, 28, 28)
Test set shape: (10000, 28, 28)
Image size: (28, 28)
Data type: uint8
Pixel value range: 0 to 255
```

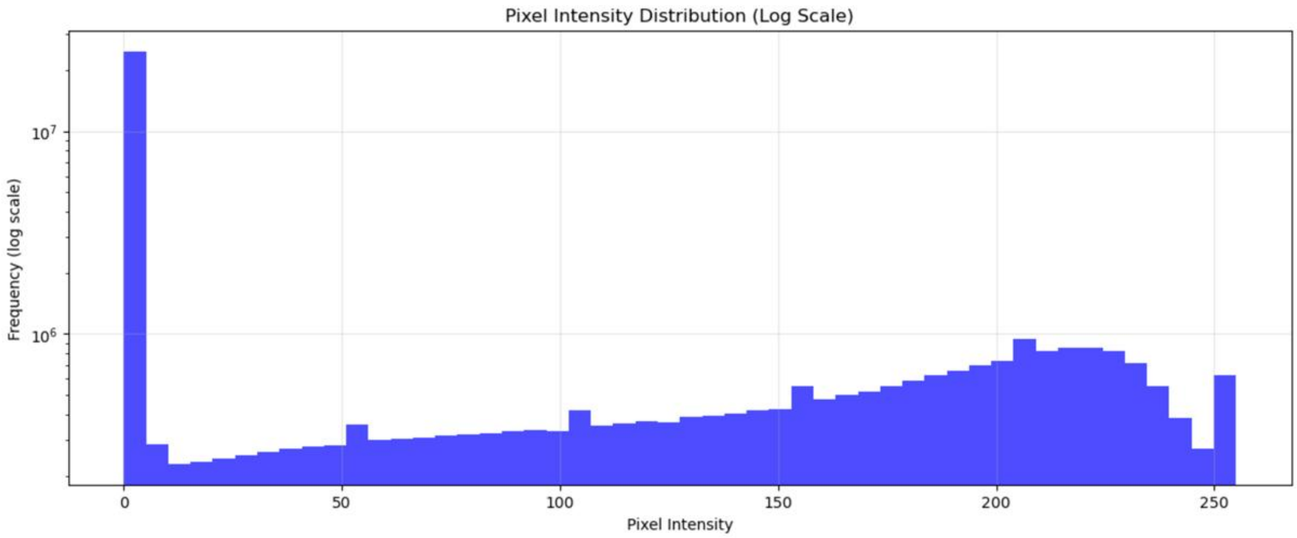
[3]:
```python
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

class_counts = np.bincount(y_train)
for i, count in enumerate(class_counts):
    print(f"{class_names[i]}: {count}")

plt.figure(figsize=(10, 5))
plt.bar(class_names, class_counts)
plt.title("Class Distribution in Training Set")
plt.xticks(rotation=45, ha='right')
plt.ylabel("Number of Samples")
plt.tight_layout()
plt.show()
```

```
T-shirt/top: 6000
Trouser: 6000
Pullover: 6000
Dress: 6000
Coat: 6000
Sandal: 6000
Shirt: 6000
Sneaker: 6000
Bag: 6000
Ankle boot: 6000
```



[4]:
```python
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(X_train[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[y_train[i]])
plt.tight_layout()
plt.show()
```

[8]:
```python
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import fashion_mnist

# Load the data
(X_train, _), (_, _) = fashion_mnist.load_data()

# Create two subplots for different views of the distribution
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

# Plot 1: Regular histogram with fewer bins
ax1.hist(X_train.ravel(), bins=50, range=(0, 255), color='blue', alpha=0.7)
ax1.set_title('Pixel Intensity Distribution')
ax1.set_xlabel('Pixel Intensity')
ax1.set_ylabel('Frequency')
ax1.grid(True, alpha=0.3)

# Plot 2: Log scale histogram with fewer bins
ax2.hist(X_train.ravel(), bins=50, range=(0, 255), color='blue', alpha=0.7)
ax2.set_yscale('log')
ax2.set_title('Pixel Intensity Distribution (Log Scale)')
ax2.set_xlabel('Pixel Intensity')
ax2.set_ylabel('Frequency (log scale)')
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Print summary statistics
print(f"Mean pixel intensity: {np.mean(X_train):.2f}")
print(f"Median pixel intensity: {np.median(X_train):.2f}")
print(f"Standard deviation: {np.std(X_train):.2f}")
```




```
Mean pixel intensity: 72.94
Median pixel intensity: 0.00
Standard deviation: 90.02
```

[5]:
```python
X_train_cnn = X_train.reshape(-1, 28, 28, 1)
X_test_cnn = X_test.reshape(-1, 28, 28, 1)
```

[6]:
```python
from tensorflow.keras import layers, models, optimizers

def resnet_block(input_data, filters, conv_size):
    x = layers.Conv2D(filters, conv_size, activation='relu', padding='same')(input_data)
    x = layers.BatchNormalization()(x)
    x = layers.Conv2D(filters, conv_size, activation=None, padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Add()([x, input_data])
    x = layers.Activation('relu')(x)
    return x

input_layer = layers.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, 3, activation='relu')(input_layer)
x = layers.MaxPooling2D(2)(x)
x = resnet_block(x, 32, 3)
x = layers.MaxPooling2D(2)(x)
x = resnet_block(x, 32, 3)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Flatten()(x)
output_layer = layers.Dense(10, activation='softmax')(x)

resnet_model = models.Model(inputs=input_layer, outputs=output_layer)
resnet_model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 28, 28, 1)	0	-
conv2d (Conv2D)	(None, 26, 26, 32)	320	input_layer[0][0]
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 13, 13, 32)	9,248	max_pooling2d[0]…
batch_normalization (BatchNormalizatio…	(None, 13, 13, 32)	128	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 13, 13, 32)	9,248	batch_normalizat…
batch_normalizatio… (BatchNormalizatio…	(None, 13, 13, 32)	128	conv2d_2[0][0]
add (Add)	(None, 13, 13, 32)	0	batch_normalizat… max_pooling2d[0]…
activation (Activation)	(None, 13, 13, 32)	0	add[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 32)	0	activation[0][0]
conv2d_3 (Conv2D)	(None, 6, 6, 32)	9,248	max_pooling2d_1[…
batch_normalizatio… (BatchNormalizatio…	(None, 6, 6, 32)	128	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 6, 6, 32)	9,248	batch_normalizat…
batch_normalizatio… (BatchNormalizatio…	(None, 6, 6, 32)	128	conv2d_4[0][0]
add_1 (Add)	(None, 6, 6, 32)	0	batch_normalizat… max_pooling2d_1[…
activation_1 (Activation)	(None, 6, 6, 32)	0	add_1[0][0]
global_average_poo… (GlobalAveragePool…	(None, 32)	0	activation_1[0][…
flatten (Flatten)	(None, 32)	0	global_average_p…
dense (Dense)	(None, 10)	330	flatten[0][0]

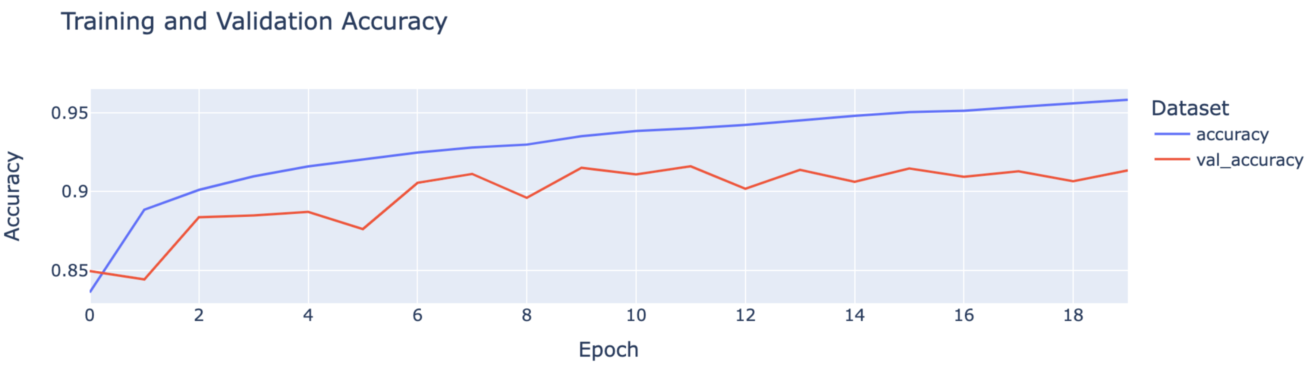
Total params: 38,154 (149.04 KB)
Trainable params: 37,898 (148.04 KB)
Non-trainable params: 256 (1.00 KB)

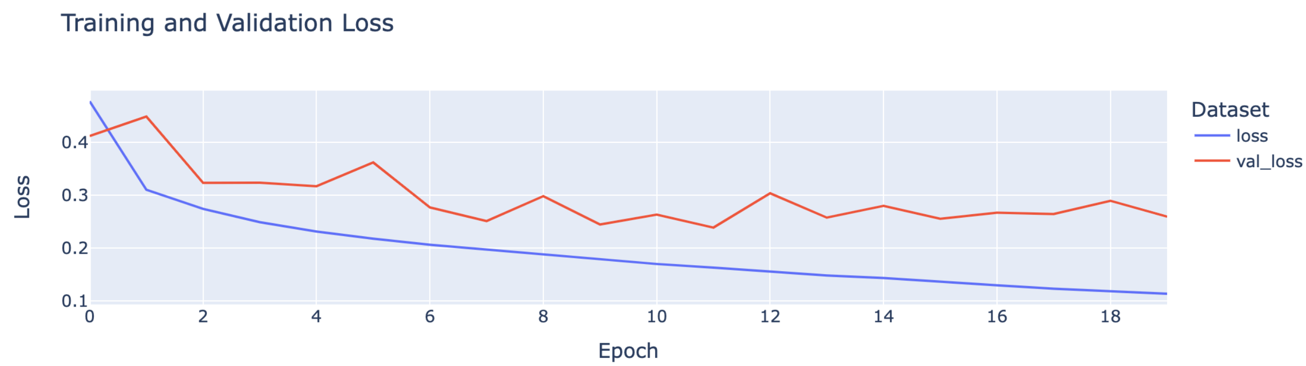
[7]:
```
resnet_model.compile(optimizer=optimizers.Adam(learning_rate=0.001),
                     loss='sparse_categorical_crossentropy',
                     metrics=['accuracy'])
```

[8]:
```
history_resnet = resnet_model.fit(X_train_cnn, y_train, epochs=20,
                                  validation_data=(X_test_cnn, y_test))
```

```
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 18s 9ms/step - accuracy: 0.7761 - loss: 0.6789 - val_accuracy: 0.8497 - val_loss: 0.4121
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 19s 10ms/step - accuracy: 0.8865 - loss: 0.3161 - val_accuracy: 0.8444 - val_loss: 0.4489
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 20s 10ms/step - accuracy: 0.8996 - loss: 0.2756 - val_accuracy: 0.8838 - val_loss: 0.3235
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 20s 11ms/step - accuracy: 0.9104 - loss: 0.2492 - val_accuracy: 0.8849 - val_loss: 0.3240
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 11ms/step - accuracy: 0.9182 - loss: 0.2262 - val_accuracy: 0.8872 - val_loss: 0.3172
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22s 12ms/step - accuracy: 0.9207 - loss: 0.2186 - val_accuracy: 0.8763 - val_loss: 0.3621
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 11ms/step - accuracy: 0.9244 - loss: 0.2070 - val_accuracy: 0.9056 - val_loss: 0.2770
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 11ms/step - accuracy: 0.9297 - loss: 0.1939 - val_accuracy: 0.9112 - val_loss: 0.2513
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 20s 11ms/step - accuracy: 0.9302 - loss: 0.1873 - val_accuracy: 0.8961 - val_loss: 0.2983
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 11ms/step - accuracy: 0.9376 - loss: 0.1762 - val_accuracy: 0.9151 - val_loss: 0.2449
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 23s 12ms/step - accuracy: 0.9404 - loss: 0.1656 - val_accuracy: 0.9109 - val_loss: 0.2635
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 11ms/step - accuracy: 0.9416 - loss: 0.1600 - val_accuracy: 0.9161 - val_loss: 0.2390
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 11ms/step - accuracy: 0.9457 - loss: 0.1490 - val_accuracy: 0.9018 - val_loss: 0.3038
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 11ms/step - accuracy: 0.9480 - loss: 0.1437 - val_accuracy: 0.9138 - val_loss: 0.2579
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22s 12ms/step - accuracy: 0.9509 - loss: 0.1382 - val_accuracy: 0.9062 - val_loss: 0.2799
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22s 12ms/step - accuracy: 0.9510 - loss: 0.1348 - val_accuracy: 0.9147 - val_loss: 0.2558
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22s 12ms/step - accuracy: 0.9543 - loss: 0.1231 - val_accuracy: 0.9094 - val_loss: 0.2672
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 24s 13ms/step - accuracy: 0.9563 - loss: 0.1178 - val_accuracy: 0.9129 - val_loss: 0.2646
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22s 12ms/step - accuracy: 0.9579 - loss: 0.1144 - val_accuracy: 0.9066 - val_loss: 0.2897
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22s 12ms/step - accuracy: 0.9613 - loss: 0.1069 - val_accuracy: 0.9135 - val_loss: 0.2596
```

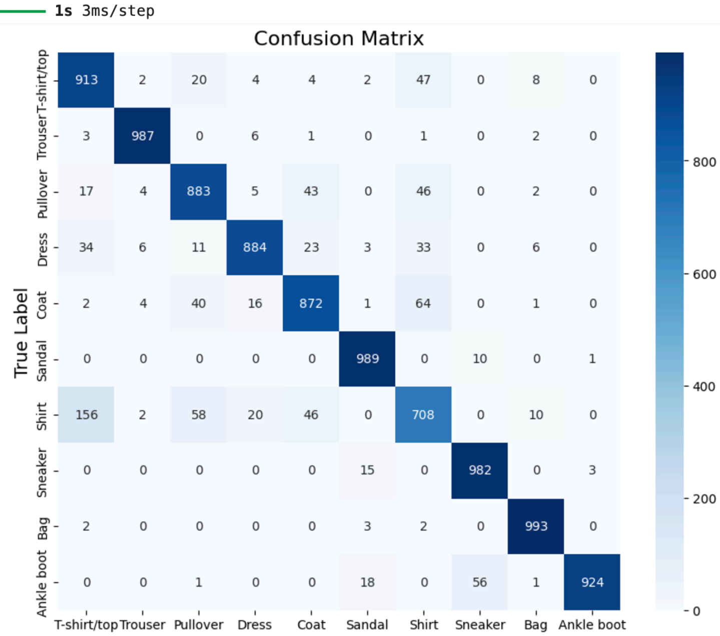
[9]:
```
# Convert history to DataFrame
history_df = pd.DataFrame(history_resnet.history)

# Plot training & validation accuracy values
fig = px.line(
    history_df,
    y=['accuracy', 'val_accuracy'],
    labels={'index': 'Epoch', 'value': 'Accuracy', 'variable': 'Dataset'},
    title='Training and Validation Accuracy'
)
fig.update_layout(
    xaxis_title='Epoch',
    yaxis_title='Accuracy',
    legend_title='Dataset',
    font=dict(size=14)
)
fig.show()

# Plot training & validation loss values
fig = px.line(
    history_df,
    y=['loss', 'val_loss'],
    labels={'index': 'Epoch', 'value': 'Loss', 'variable': 'Dataset'},
    title='Training and Validation Loss'
)
fig.update_layout(
    xaxis_title='Epoch',
    yaxis_title='Loss',
    legend_title='Dataset',
    font=dict(size=14)
)
fig.show()
```






```python
[10]: from sklearn.metrics import confusion_matrix, classification_report

# Predict on test data
y_pred_resnet = np.argmax(resnet_model.predict(X_test_cnn), axis=1)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_resnet)

# Plot confusion matrix using Seaborn heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix', fontsize=16)
plt.xlabel('Predicted Label', fontsize=14)
plt.ylabel('True Label', fontsize=14)
plt.show()

# Classification Report
print('Classification Report')
print(classification_report(y_test, y_pred_resnet, target_names=class_names))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```



Predicted Label

```
Classification Report
              precision    recall  f1-score   support

 T-shirt/top       0.81      0.91      0.86      1000
     Trouser       0.98      0.99      0.98      1000
    Pullover       0.87      0.88      0.88      1000
       Dress       0.95      0.88      0.91      1000
        Coat       0.88      0.87      0.88      1000
      Sandal       0.96      0.99      0.97      1000
       Shirt       0.79      0.71      0.74      1000
     Sneaker       0.94      0.98      0.96      1000
         Bag       0.97      0.99      0.98      1000
  Ankle boot       1.00      0.92      0.96      1000

    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000
```